Jennrich
Massmann
Schulz

3-D-Grafik-programmierung

Ein **DATA BECKER** Buch



AMIGA

Jennrich
Massmann
Schulz

3-D-Grafik-programmierung

Ein **DATA BECKER** Buch



AMIGA

Vorwort

Das erste Mal, daß ich je etwas von Computergrafik gehört und gesehen habe, war zu Beginn meines Informatikstudiums in Dortmund, als im Fernsehen ein Bericht über den damaligen Entwicklungsstand auf diesem Gebiet gesendet wurde. Gezeigt wurde unter anderem ein japanischer Computerkurzfilm, in dem ein aus verchromten, mechanischen Teilen zusammengesetzter Tiger zu sehen war der über eine verspiegelte Ebene trabte. Ich war sofort fasziniert von diesen Bildern. Heute sind sie für uns fast alltäglich; fast jede Fernsehsendung, die etwas auf sich hält, hat in ihrem Vorspann einen computerberechneten Film.

Einige Wochen später lernte ich Peter Schulz kennen, der schon damals an einem Computergrafikprogramm bastelte, das jedoch auf dem Sinclair Spectrum eine geschätzte Rechenzeit von einem halben Jahr für ein Bild gebraucht hatte. Im Laufe der Zeit wurde Peters Computerarsenal immer größer, die Computer immer leistungsfähiger, teurer und vor allem schneller. Infolgedessen schrumpfte die Rechenzeit schon bald auf einige Monate, dann auf ein paar Wochen, schließlich auf wenige Tage zusammen.

Entscheidend hierfür waren nicht nur die schnelleren Rechner, sondern auch die verbesserten und optimierten Algorithmen zur Bildberechnung. Als am Anfang dieses Jahres ein Bild in sage und schreibe 20 Stunden fertig war, entstand schließlich die Idee zu dem nun vorliegenden Buch.

In diesem Buch soll Ihnen nicht nur eine komplette Lösung des Problems der computerberechneten Grafik präsentiert werden, sondern besonders die Grundlagen des Ray-Tracings (so heißt das verwendete Berechnungsverfahren in der Fachsprache), die hierfür benötigten mathematischen Grundlagen und die Entwicklung aller notwendigen Algorithmen. Sie sollen (so hoffen wir) nach der Lektüre unseres Buches in der Lage sein, ein auf Ihre Ansprüche maßgeschneidertes Programm selbst zu entwickeln oder das von uns veröffentlichte so zu verändern, wie Sie sich es wünschen.

Da die Arbeit für zwei mitten in ihrem Studium steckende Autoren in so kurzer Zeit nicht zu bewältigen gewesen wäre, stieß noch ein Dritter zu uns: Bruno Jennrich, der genauso wie wir (Peter und Andreas) viel Arbeit, Zeit und Ideen in dieses Projekt steckte.

An dieser Stelle möcht ich mich noch ganz herzlich bei folgenden Leuten bedanken:

Oli, der mir in meiner computerlosen Zeit mit seinem Amiga über die Runden half; Siggi, bei dem ich meterlange Programmausdrucke habe anfertigen lassen und dessen Druckersicherung fast immer dabei durchbrannte; Petra, Christian und alle anderen, die in letzter Zeit leider viel zu viel über dieses Buch gehört haben.

Andreas Massmann *Oberhausen, Mai 1987*

Inhaltsverzeichnis

1.	**Einleitung**	**13**
2.	**Die Diskette zum Buch**	**15**
3.	**Grundlagen des Ray-Tracing**	**17**
3.1	Von der Natur	18
3.2	In den Computer	18
3.2.2	Schatten	20
3.2.3	Spiegelung	21
3.2.4	Transparenz	21
3.2.5	Grenzen des Verfahrens	22
3.3	Darstellung im Computer	23
3.3.1	Datenstruktur	32
3.4	Simulation der Wahrnehmung	36
3.4.1	Sehen wir was?	40
3.4.2	Was sehen wir?	55
3.4.3	Die Schattenseiten der Welt	63
3.4.4	Spieglein, Spieglein	66
3.4.5	Maler Klecksel oder: Wie bringe ich die Farbe auf den Bildschirm	69
3.5	Optimierungen	73
4.	**Realisierung des Tracers**	**83**
4.1	Mehr dreidimensionale Grafik!	83
4.1.1	Von 3-D nach 2-D	83
4.1.2	Körper und Flächen	99
4.2.	Der Editor	115
4.2.1	Warum überhaupt ein Editor	115
4.2.2	Aufgaben und Einzelheiten des Editors	116

4.2.3 Die einzelnen Komponenten des Editors 122
4.2.3.1 Die Eingabe 123
4.2.3.2 Die Ausgabe 135
4.2.3.3 Die Grafik 138
4.2.3.4 Die Datenoperationen 143
4.2.3.5 Die Diskettenoperationen 150
4.2.3.6 Die Maus 152
4.2.4 Wir bauen uns einen Editor 159
4.3 Das Hauptprogramm 160
4.3.1 Die restlichen Routinen des Tracers 160
4.3.2 Das Hauptprogramm 209

5. Die Bedienung der Programme 211

5.1 Handbuch zum Editor 211
5.1.1 Allgemeines zur Eingabe 211
5.1.2 Die Eingabe der Körperdaten 213
5.1.3 Die weiteren Operationen 219
5.1.4 Der Materialeditor 221
5.1.5 Die Transformationen 222
5.1.6 Die Diskettenoperationen 223
5.2 Bedienung des Tracers 224
5.2.1 Nach dem Start 224
5.2.2 Das 'Amiga'-Menü 225
5.2.3 Das 'Datei'-Menü 225
5.2.3.1 Laden 226
5.2.3.2 Materialien 228
5.2.3.3 Hintergrund 228
5.2.3.4 Bilder abspeichern 230
5.2.3.5 Bilder ausdrucken 230
5.2.3.7 Das Programm verlassen 231
5.2.4 Das 'Editor'-Menü 232
5.2.5 Das 'Parameter'-Menü 232
5.2.6 Das 'Zeichne'-Menü 234
5.2.6.1 Schattieren 234
5.2.6.2 Das Drahtmodell 236
5.2.7 Steuerung des Programms über Tastatur 237

6. Ausblicke **239**

7. Mathematische Grundlagen **247**

7.1 Vektorrechnung 247
7.2 Was man mit Vektoren machen kann 250

8. Tips und Tricks zur Bilderstellung **261**

Anhänge **267**

Anhang A: Der Modularaufbau des Tracers 267
Anhang B: Fehlermeldungen des Tracers 273

Stichwortverzeichnis **275**

1. Einleitung

Kennen Sie das auch? Da haben Sie sich ein schlaues Buch über ein mathematisches Thema gekauft, weil Sie Ihr Wissen erweitern wollen, doch beim Lesen stellen Sie fest, daß sich zwar alles ganz faszinierend anhört, aber, ganz ehrlich, verstanden haben Sie (zu) wenig.

Auch dieses Buch hat einen mathematischen Sachverhalt zum Inhalt, allerdings möchten wir, daß Sie unsere Erklärungen auch wirklich verstehen. Darum gibt es neben den Beispielen und Erklärungen das Programm selbst. Da die Algorithmen nicht nur formelmäßig vorhanden sind, sondern auch als funktionstüchtiges Programm, ist es für Sie einfacher, den Algorithmus nachzuvollziehen.

Ray-Tracing, wie der Algorithmus genannt wird, um den sich in diesem Buch alles dreht, beruht nun leider auf einigen mathematischen Sachen wie Vektorrechnung und Determinanten. Trotzdem ist das Grundprinzip des Algorithmus verhältnismäßig einfach, so daß Sie auch ohne große mathematische Kenntnisse in der Lage seien dürften, dieses zu verstehen.

Wenn Sie allerdings das Programm nachvollziehen wollen, so müssen Sie sich schon etwas besser in der Mathematik auskennen. Alles, was Sie brauchen, finden Sie in den mathematischen Grundlagen kurz erklärt. Die behandelten Themen sollten Ihnen größtenteils auf der Schule beigebracht worden sein, zumindest im Leistungskurs Mathematik, sofern Sie einen solchen belegt hatten. Ein Mathematik-Studium ist also nicht Voraussetzung für das Verstehen dieses Buches.

2. Die Diskette zum Buch

Auf der beiliegenden Diskette finden Sie nebem dem Unterverzeichnis 'Modules', das die einzelnen Programmmodule enthält, auch das Verzeichnis 'Bilder'. Sie erhalten dort eine kleine Kostprobe von dem, was Sie erwartet.

'Bilder' enthält einige Bilder - abgespeichert im IFF-Format, die zeigen, zu welchen fantastischen Bildern Sie nach der Lektüre dieses Buches fähig sein werden.

Booten Sie die beigelegte Diskette. Stecken Sie die Diskette also ins Laufwerk 'df0:', nachdem der Rechner die 'Workbench' verlangt. Nun werden Ihnen nacheinander drei Bilder gezeigt. Um die Darstellung eines Bildes zu beenden, brauchen Sie nur die linke Maustaste zu drücken, wenn der Mauszeiger in der linken oberen Ecke steht. Dann wird das nächste Bild gezeigt.

Die Routine, die die IFF-Bilder einliest und darstellt ('ZeigILBM' im Verzeichnis 'c'), wurde in 'C' geschrieben. Das Source-Listing dieser Routine befindet sich im Unterverzeichnis 'Modules'. Sie liest im übrigen alle IFF-Bilder (auch 'gepackte').

Bitte haben Sie Verständnis dafür, daß wir nicht mehr als drei Bilder auf die Diskette kopieren konnten. Der 'Editor' und das Unterverzeichnis 'Libs', in dem alle benötigten '.bmap'-Files enthalten sind, müssen ja auch auf der Diskette Platz finden. So kann das Programm nahezu unabhängig von AmigaBASIC laufen. Außerdem mußte auch der 'Tracer' auf die Diskette.

Das Maschinensprache-Programm 'SetPoint.B' sowie das Source-File 'SetPoint.ASM' im Modules-Verzeichnis werden auch benötigt.

Sind Sie jedoch an weiteren Bildern interessiert, können Sie eigene erstellen oder im Rahmen der 'Public-Domain'-Software in Umlauf geratene Bilder sammeln.

Doch kommen wir zu dem letzen Unterverzeichnis der Diskette. Im 'Objekte'-Unterverzeichnis finden Sie einige Objektdefinitionen sowie die dazugehörigen Materiallisten einiger Objekte. Das schattierte Ergebnis dreier dieser Objekte finden Sie im 'Bilder'-Verzeichnis. Weitere sind in diesem Buch abgebildet.

3. Grundlagen des Ray-Tracing

Computergrafik ist eine feine Sache. Besonders faszinierend ist es jedoch, wenn der Computer Daten oder Gegenstände dreidimensional darstellt. Als einfachste Möglichkeit gibt es zum Beispiel Netzgrafiken, die nicht besonders viel Rechenaufwand erfordern. Sollen dabei allerdings nur die sichtbaren Linien gezeichnet werden, so werden meist sogenannte Hidden-Line-Algorithmen (Hidden-Line = verdeckte Linien) angewendet.

Die Ergebnisse, die man mit Hidden-Line-Algorithmen erzielen kann, sind schon recht gut. Schwierig wird es erst, wenn sich Objekte schneiden oder zyklisch verdecken. Zwar gibt es auch dafür Lösungen, wie die Aufteilung der Objekte in kleinere, aber dies soll nicht Thema dieses Buches sein.

Wenn man einen Schritt weiter in Richtung realitätsnaher Darstellung geht, so kommt man zu den Hidden-Surface-Algorithmen (Hidden-Surface = verdeckte Flächen). Das darzustellende Objekt wird dabei zumeist in viele kleine Dreiecke zerlegt. Um nun eine Darstellung mit verdeckten Flächen zu bekommen, kann man diese beispielsweise sortieren und zwar so, daß die Dreiecke, die am weitesten hinten liegen, zuerst gezeichnet werden, und diejenigen, die am weitesten vorne liegen, zuletzt. Wenn die Dreiecke nur klein genug gewählt werden, so erhält man eine recht gute Darstellung. Färbt man die Dreiecke noch in Abhängigkeit von ihrem Winkel zur Lichtquelle verschieden hell ein (fällt das Licht senkrecht auf ein Dreieck, so ist dies hell, fällt es in einem flachen Winkel auf dessen Oberfläche, so ist dies dunkel), so entsteht eine ansprechende Darstellung.

Solche Hidden-Surface-Algorithmen sind recht schnell und daher besonders für Homecomputer geeignet, weswegen diese auch schon öfter Thema von Büchern waren. Wir wollen aber noch einen Schritt weiter gehen, nämlich zum Ray-Tracing.

Ray-Tracing ist zwar langsamer als die oben genannten Algorithmen, bietet dafür aber wesentliche Vorteile, wie zum Beispiel einfache Darstellung von Schatten, Spiegelung oder Transparenz, welche einem Bild erst den letzten Schliff geben.

Außerdem ist der Grundgedanke von Ray-Tracing recht einfach, so daß Sie nicht gleich Mathematik studieren müssen, um ihn zu verstehen.

3.1 Von der Natur

Bevor wir daran gehen, einen Ray-Tracing-Algorithmus zu entwickeln, sehen wir uns einmal an, wie der Wahrnehmungsvorgang in der Natur vor sich geht.

Die Sonne sendet Lichtstrahlen aus, welche, nachdem sie eine Weile durch den Raum geflogen sind, irgendwann mal auf einen Gegenstand treffen, zum Beispiel auf einen grünen Würfel. Doch halt! Warum ist der Würfel grün? Richtig, er absorbiert alle nicht-grünen Anteile des Lichtstrahls, und er reflektiert alle grünen Anteile.

Wenn dieser Lichtstrahl in unser Auge trifft, so sehen wir einen grünen Punkt, weil der Lichtstrahl eben grün ist, in genau der Richtung, in der auch der Würfel liegt. Da die Sonne (Lichtquelle) nun aber ziemlich viele Lichtstrahlen ausschickt, treffen eine ganze Menge den Würfel und ebenfalls nicht gerade wenige gelangen, nachdem sie vom Würfel reflektiert worden sind, in unser Auge, so daß dort aus der Summe der einzelnen Lichtstrahlen das Bild eines Würfels entsteht.

3.2 In den Computer

Um diesen Vorgang mit dem Computer nachzuvollziehen, müssen wir ein paar Einschränkungen machen. Zum einen besitzt Ihr Computer nur ein Auge, was im wesentlichen dadurch bedingt ist, daß er auch nur einen Bildschirm hat. Außerdem ist die Lichtquelle, von der die künstliche Szene beleuchtet wird, punktförmig. Ein wesentlich einschneidenderes Problem gibt es allerdings noch:

In der Realität sendet die Lichtquelle Lichtstrahlen aus, und zwar ausgesprochen viele; so viele, daß selbst alle Computer zusammen, die es auf diesem Planeten gibt, wohl nicht ausreichen würden, diesen Vorgang in einer akzeptablen Zeit zu simulieren. Außerdem wäre dieses Verfahren reichlich uneffizient, da sehr viele Lichtstrahlen nie auf einen Gegenstand treffen würden.

Drehen wir den Spieß also um: Unser Auge sendet Sehstrahlen aus, die dann zum Beispiel wieder auf den grünen Würfel treffen. Dort angekommen "überlegt" sich der Sehstrahl, welche Farbe der Würfel hat, und meldet diese Information unmittelbar an das Auge zurück. Um nochmals auf die Vereinfachungen zurückzukommen, nehmen wir ferner an, daß unsere Netzhaut rechteckig ist und daß die Sehzellen, die sich dort befinden, allesamt Farbe empfangen können.

Sie merken schon: Die Netzhaut des Auges entspricht dem Bildschirm des Computers. Alles, was unser Computer nun zu tun hat, ist, für jeden Pixel des Bildschirms... pardon, für jede Sehzelle der Netzhaut einen Sehstrahl loszuschicken und zu prüfen, ob dieser einen Gegenstand trifft, und wenn ja, dessen Farbe zu bestimmen und auf dem Bildschirm anzuzeigen. Die Richtung, in die die Sehstrahlen ausgesendet werden, ist ebenfalls festgelegt, da diese unser Auge nur durch die Linse verlassen können. Idealerweise müssen diese alle die Linse exakt in der Mitte durchqueren; wir könnten aber auch von einer punktförmigen Linse ausgehen.

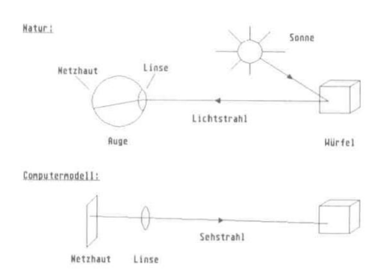


Abbildung 3.1

3.2.1 Licht

So, jetzt sehen wir also einen Würfel, oder vielleicht nicht? Eigentlich sehen wir bis jetzt nur einen grünen Umriß von etwas, was ein Würfel sein könnte. Was noch fehlt, ist eine Schattierung, die den Würfel plastisch macht. Die Seiten des Würfels, auf die die Lichtquelle senkrecht strahlt, sind heller, als die, die gar nicht beleuchtet werden. Für den Computer ist es nicht übermäßig schwer zu bestimmen, wie hell eine Seite ist: Je mehr die Lichtquelle senkrecht über der Fläche steht, desto heller ist die Seite.

Mit derselben Methode kann man auch herausfinden,. ob die Seite des Gegenstands, die man sieht, überhaupt von der Lichtquelle angestrahlt wird, oder ob man die "Rückseite" sieht. Darauf wird aber erst später eingegangen, wenn wir ins Detail gehen.

3.2.2 Schatten

Richtig interessant wird es erst, wenn der Computer auch noch den Schatten berechnen kann. In der Realität ist überall dort Schatten, wo kein Licht hinfällt. Da wir in unserem Simulationsmodell aber nicht von Lichtstrahlen, sondern von Sehstrahlen ausgehen, müssen wir die Sache etwas anders anpacken. Stellen wir uns für einen Augenblick vor, wir wären die Lichtquelle. Dann wären unsere Sehstrahlen eigentlich nichts anderes als die Lichtstrahlen der Lichtquelle. Überall dort, wo wir hinschauen könnten, wäre Licht, und an den übrigen Stellen, wo wir nicht hinsehen könnten, wäre Schatten.

Doch nun wieder kehrt um zur Ausgangslage: Unser Auge hat seinen Sehstrahl losgeschickt und überlegt nun, wie hell dieser Punkt des Würfels ist. Liegt dieser im Schatten, so ist er dunkel. Nun drehen wir den Spieß wieder um. Der Sehstrahl schaut jetzt von dem Punkt auf dem Würfel in Richtung der Lichtquelle. Wenn er die Lichtquelle nicht sehen kann, so befindet er sich im

Schatten (alte Bauernweisheit: Kannst du die Sonne nicht sehen, so mußt du wohl im Schatten stehen) und damit auch der Punkt des Würfels.

3.2.3 Spiegelung

Jetzt machen wir mit der Spiegelung weiter. Wie wir schon ganz zu Anfang gesehen haben, wird in der Realität stets ein Teil der Lichtstrahlen von einer Oberfläche reflektiert und ein Teil absorbiert. Der reflektierte Teil kann nun seinerseits wieder auf eine andere Oberfläche treffen, und von dieser wieder reflektiert werden, solange, bis er irgendwann in unser Auge trifft oder völlig absorbiert worden ist.

Bei unseren Sehstrahlen gehen wir analog vor. Stellt der Sehstrahl fest, daß die Oberfläche, auf die er gestoßen ist, verspiegelt ist, so merkt er sich deren Farbe, Helligkeit und den Grad der Verspiegelung. Dann läßt er sich von der Oberfläche reflektieren zu einem Gegenstand, der in seiner Richtung liegt. Auch dort stellt er wieder Farbe, Helligkeit und Spiegelung fest und läßt sich weiter reflektieren, vorausgesetzt, diese Oberfläche war ebenfalls verspiegelt.

Dieses wiederholt sich nun solange, bis der Sehstrahl einen Gegenstand trifft, der überhaupt nicht verspiegelt ist, also eine völlig stumpfe Oberfläche hat, oder bis er keinen Gegenstand mehr findet. Dann bildet er aus allen Farben, Helligkeiten und Spiegelungsfaktoren eine Farbe und eine Helligkeit, die er dem Auge mitteilt. Darauf, wie diese ganzen Werte zu einem einzigen verknüpft werden, werden wir später noch im Detail eingehen. Das Verfahren ist aber dem der Realität nachempfunden.

3.2.4 Transparenz

Für ganz Unerschrockene sei zuletzt noch die Transparenz erwähnt. Diese ist eigentlich nichts anderes als die Spiegelung, nur mit dem Unterschied, daß der Licht- bzw. Sehstrahl nicht von der Oberfläche reflektiert wird, sondern durch diese hindurch-

tritt. Hat unser Sehstrahl also festgestellt, daß eine Oberfäche transparent ist, so setzt er seinen Weg in die Richtung fort, in die er sich auch bewegt hat, bevor er auf diese Oberfläche gestoßen ist.

Wenn eine Oberfäche transparent und verspiegelt ist, so muß sich der Sehstrahl teilen oder einen rekursiven Algorithmus anwenden, da er ja nicht gleichzeitig durch die Oberfläche hindurch treten und von dieser reflektiert werden kann. Während sich in der Realität der Lichtstrahl teilt, wird unserer Computer mit einem rekursiven Algorithmus wohl besser bedient sein, oder wollen Sie Ihren Computer teilen?

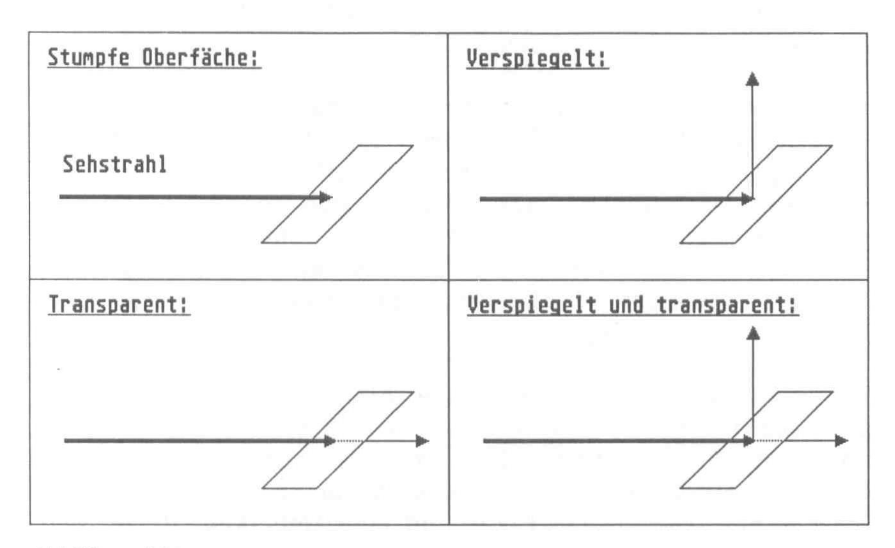


Abbildung 3.2

3.2.5 Grenzen des Verfahrens

Unser Algorithmus ist in der Lage, Schattierung, Schatten, Spiegelung und Transparenz zu simulieren. Allerdings mit Einschränkungen: Da wir mit Sehstrahlen statt mit Lichtstrahlen arbeiten, können wir nicht simulieren, wie Licht in einem Spiegel gespiegelt wird. Was unseren Bildern, die wir errechnen lassen

wollen, fehlt, sind die Effekte, die durch Streulicht entstehen, also indirekte Beleuchtung oder Lichtbrechungen.

Ein weiteres Problem ergibt sich dadurch, daß Lichtstrahlen nicht aus einer einzigen Farbe bestehen, sondern aus vielen verschiedenen, deren Summe die Farbe ergibt, in der wir das Licht wahrnehmen. Nicht umsonst spricht man von Spektrum des Lichts. Bei der Reflektion von Lichtstrahlen wird ein Teil dieses Spektrums absorbiert, und der Rest fliegt unbehelligt weiter. In unserem Simulationsmodell werden wir aber von einem Lichtstrahl mit festgelegter Farbe ausgehen. Allerdings sind die Nachteile dieser Methode bei weitem nicht so gravierend wie die zuvor genannten.

3.3 Darstellung im Computer

Hinweis: Da nun im weiteren Verlauf des Buches häufig mathematische Ausdrücke und Formulierungen gebraucht werden, bitten wir Sie, im Bedarfsfall im Kapital 7 nachzulesen, was gemeint ist. Aber nun weiter.

Nachdem der grundsätzliche Algorithmus umgangssprachlich formuliert ist, wollen wir uns nun daran machen, ihn unserem Computer verständlich zu machen. Vorher müssen wir aber erst mal etwas schaffen, was er dann "sehen" kann.

Da der Computer eigentlich "nur" rechnen kann, liegt es nahe, die Welt des Computers mathematisch zu beschreiben. Naürlich fällt Ihnen dazu sofort dreidimensionale Geometrie ein. Mit relativ einfachen Formeln kann man Geraden, Ebenen und sogar Kugeln definieren. Eine Ebene ist wie folgt bestimmt:

```
r = r1 + u*r2 + v*r3
```

wobei r, r1, r2 und r3 Vektoren und u und v Skalare sind. r1 ist dabei ein beliebiger Punkt, von dem wir wissen, das er auf der Ebene liegt. Durch r2 und r3 wird dann die Ebene aufgespannt, wobei wir allerdings darauf achten müssen, daß r2 und r3 linear

unabhängig sind. Setzt man nun für u und v alle möglichen Werte ein, so erhält man alle möglichen Punkte, die auf der Ebene liegen.

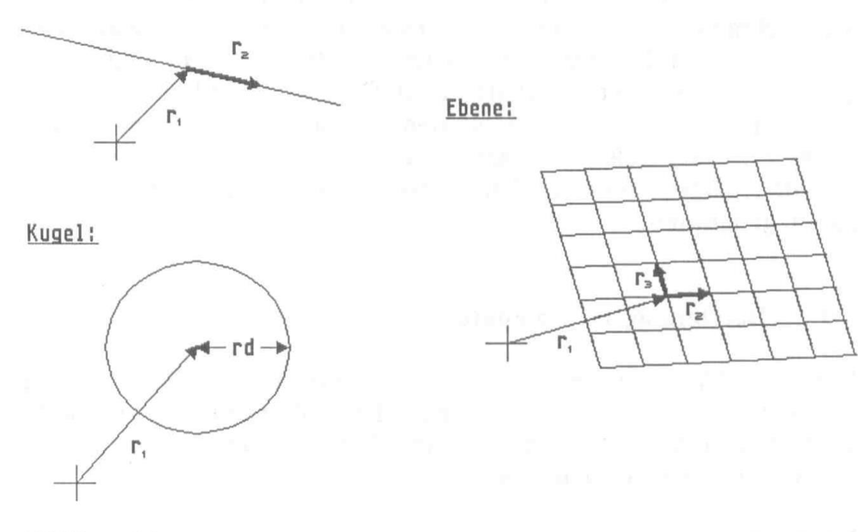


Abbildung 3.3

Diese unsere Ebene hat nur einen Nachteil: sie ist unendlich. Dadurch eignet sie sich zwar vorzüglich für den Boden unserer Welt, aber einen Würfel können wir damit nicht definieren. Wir müssen die Ebene irgendwie einschränken. u und v dürfen bei der Ebene normalerweise beliebig große Werte annehmen. Es liegt also nahe, den Definitionsbereich für u und v zu begrenzen, sagen wir auf [0,1]. u und v müssen also größer als Null und kleiner als Eins sein. Wenn wir nun wieder für u und v alle erlaubten Werte einsetzen, so stellen wir fest, daß alle Punkte zusammen ein Parallelogramm bilden.

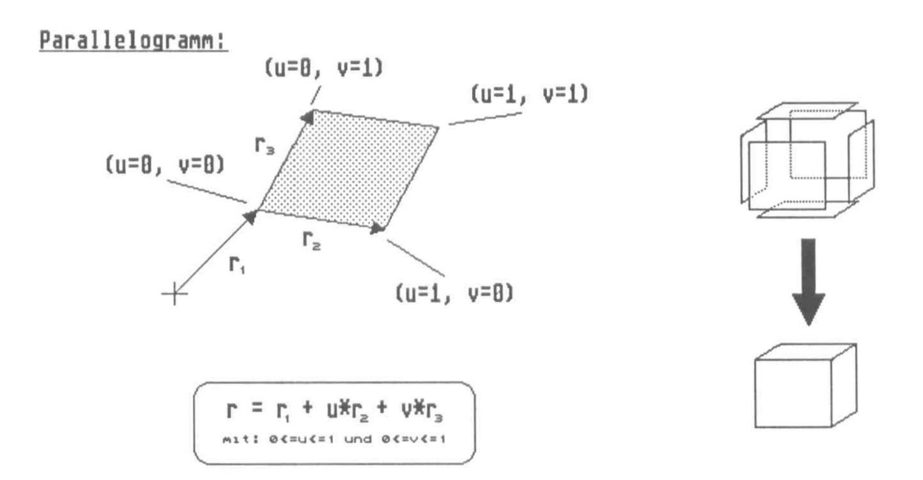


Abbildung 3.4

Wählen wir r2 und r3 so, daß diese senkrecht aufeinander stehen, so erhalten wir ein Rechteck, das nichts anderes als ein Spezialfall des Parallelogramms ist. Wenn dann die Längen von r2 und r3 auch noch gleich groß sind, so haben wir ein Quadrat, und mit sechs Quadraten können wir dann unseren Würfel zusammenbauen (Bild 3.4).

Nun ist aber das Parallelogramm noch nicht der Weisheit letzter Schluß. Was wir unbedingt noch brauchen sind Dreiecke, denn jeder Körper läßt sich letztendlich in Dreiecke zerlegen. Zu diesem Zweck zeichnen wir ein Koordinatensystem und darein die Funktion $y = -x + 1$. Die Fläche, die durch die Funktionsgerade und die beiden Achsen eingeschlossen wird, sieht einem Dreieck schon sehr ähnlich. Wenn wir die Formel etwas umformen, so erhalten wir $x + y = 1$. Aber wo nehmen wir x und y her?

Die Achsen des Koordinatensystems sehen mit ihren Pfeilen an den Enden schon wie Vektoren aus. Sei die X-Achse der Vektor r2 und die Y-Achse der Vektor r3. Genauer gesagt sei der Einheitsvektor der X-Achse gleich r2 und der Einheitsvektor der

Y-Achse r3. Ferner ersetzen wir in der Formel x durch u und y durch v. Da wir eine Dreiecksfläche haben wollen und nicht nur einen Umriß, ersetzen wir das Gleichheitszeichen durch ein "<=". Wir erhalten damit:

```
u + v <= 1
```

Außerdem sollen u und v beide positiv sein, sonst würden wir wieder eine unendliche Fläche erhalten. Dies nehmen wir als Einschränkung für unsere Ebene. Setzen wir jetzt wieder alle erlaubten Werte für u und v in die Ebenengleichung ein, so liegen alle Punkte innerhalb des Dreiecks, von dem r2 und r3 zwei Seiten sind.

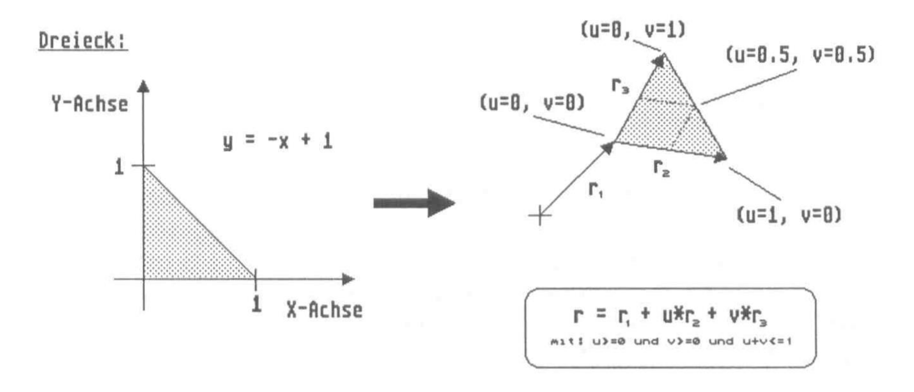


Abbildung 3.5

Mit Dreieck und Parallelogramm können wir nun bereits eine ganze Menge anfangen. Wenn wir die Gegenstände, die wir darstellen wollen, nur in ausreichend viele kleine Dreiecke zerteilen, so haben wir bereits alles an Grundobjekten, was wir brauchen. Einziger Nachteil: Um komplizierte Gegenstände durch Dreiecke darzustellen, brauchen wir eine Unmenge davon, und diese verschlingen eine Unmenge von Speicher. Da wir noch nicht soweit sind, den Arbeitsspeicher unseres Computers nach Gigabyte zu zählen, wollen wir uns ein paar Gedanken darüber machen, was wir noch an Grundobjekten definieren können.

Bevor wir uns der Kugel zuwenden, wollen wir noch ein bißchen mehr aus der Ebene herausholen. Sie erinnern sich doch sicherlich an die allgemeine Kreisgleichung:

```
x2 + y2 = rd2
```

wobei rd der Radius ist. Für den Einheitskreis liest sich die Formel noch einfacher: x2 + y2 = 1, da der Einheitskreis ja bekanntlich den Radius Eins hat. Jetzt nochmal das gleiche wie bei dem Dreieck: r2 sei der Einheitsvektor der X-Achse, r3 der Einheitsvektor der Y-Achse, u sei x und v sei y (Lassen Sie sich bloß kein x für ein u vormachen!). Die Formel wird zu:

```
u2 + v2 <= 1.
```

Diesmal müssen wir nicht fordern, daß u und v beide positiv sein sollen, da dafür schon die Quadratbildung sorgt. Setzen wir jetzt wieder alle erlaubten Werte für u und v ein, so erhalten wir eine Kreisfläche.

Kreis:

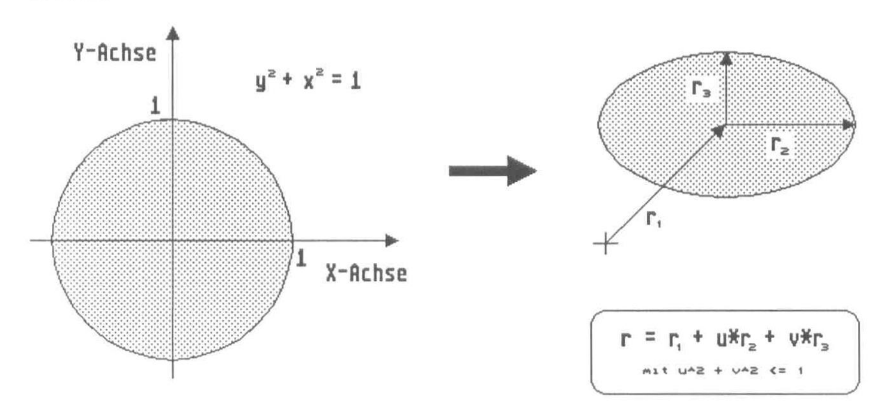


Abbildung 3.6

Sicherheitshalber sollten r2 und r3 so gewählt sein, daß diese senkrecht aufeinander stehen. Zwar ist die Gleichung nicht daran gebunden, das Ergebnis ist dann aber anschaulicher.

Haben r2 und r3 nicht die gleiche Länge, so erhalten wir eine Ellipse. Dadurch, daß wir Kreisflächen nicht mehr durch lauter kleine Dreiecke darstellen müssen, sparen wir einiges an Speicher.

Wenn wir später dreidimensionale Schrift darstellen wollen, so benötigen wir noch Kreisringe (fürs o), und für Tortendiagramme brauchen wir Kreisausschnitte. Diese erhalten wir beide über die Kreisfläche. Beim Kreisring berechnen wir zuerst die Länge des Vektors (u,v), also

```
l = u2 + v2
```

und fordern dann, daß l in einem bestimmten Intervall, also zwischen zwei Werten, liegen muß. Je nachdem, wie wir diese Werte, die Intervallgrenzen, wählen, erhalten wir einen dicken oder einen dünnen Kreisring.

Für den Kreisausschnitt müssen wir ein bißchen mehr tun. Zusätzlich zur Länge l berechnen wir den Winkel w, der zwischen (u,v) und der r2-Achse liegt. Im Grunde genommen berechnen wir die Polarkoordinaten von (u,v). Ebenso wie wir für den Kreisring l eingeschränkt haben, schränken wir nun w ein, zum Beispiel auf [0,1.57]=> 0<=w<=1.57, was einem Viertelkreis entspricht.

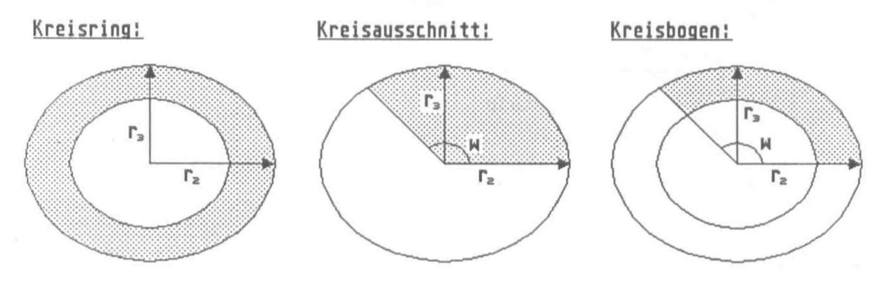


Abbildung 3.7

Zum Schluß können wir natürlich auch w und l einschränken, so daß wir einen Kreisbogen erhalten.

Doch jetzt endlich zur Kugel. Brauchen wir sie wirklich, wo wir doch alle möglichen Objekte in Dreiecke aufteilen können? Die Antwort heißt ja! Um einen Kreis aus Dreiecken zusammenzusetzen, so daß man die einzelnen Dreiecke nicht mehr erkennt, braucht man 50 bis 100 Stück. Für eine Kugel bräuchten wir dementsprechend 2*502 bis 2*1002, also 5000 bis 20000 Dreicke. Eine richtige Kugel hat zudem den Vorteil, daß sie immer eine völlig runde Oberfläche hat und man auch bei starker Vergrößerung keine Ecken und Kanten sieht.

Die allgemeine Kugelgleichung lautet:

```
(r - r1)2 = rd2
```

wobei r1 der Mittelpunkt der Kugel und rd deren Radius ist. Mehr kann man zur Kugel eigentlich nicht sagen, da es etwas schwierig ist, irgendwelche Einschränkungen vorzunehmen. Belassen wir es also bei der Kugel als Kugel und versuchen nicht, daraus weitere Objekte zu gewinnen.

Und nun auf zu weiteren Grundobjekten. Wie wäre es mit Zylinder, Kegel und Ellipsoid? Ganz so einfach wie die bisherigen Objekte lassen sich diese nicht darstellen, deshalb greifen wir zu einem Trick.

Bisher haben wir zusätzliche Objekte dadurch erhalten, daß wir bei unseren bereits vorhandenen Objekten Einschränkungen vorgenommen haben. Bei der Kugel war dies nicht möglich, da dort außer Mittelpunkt und Radius keine weiteren Parameter vorhanden waren. Nach unserer zweidimensionalen Fläche definieren wir uns jetzt einen dreidimensionalen Raum:

```
r = r1 + u*r2 + v*r3 + w*r4
```

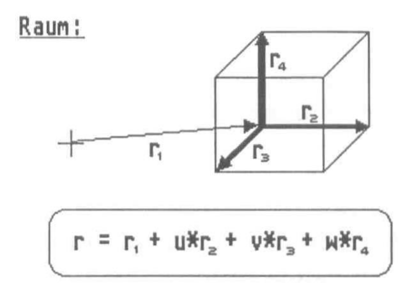
wobei r, r1, r2, r3 und r4 Vektoren und u, v und w Skalare sind. Außerdem sollten r2, r3 und r4 senkrecht aufeinander stehen, weil dies anschaulicher ist, und r2, r3 und r4 dürfen nicht komplanar sein, da sie sonst keinen Raum aufspannen, sondern nur eine Ebene.

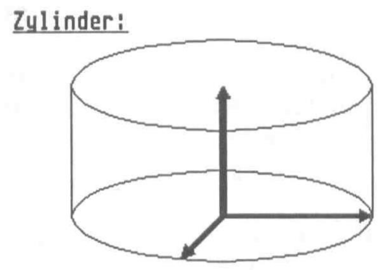
Legen wir keine Einschränkung für u, v und w fest, so ergibt dies keinen Sinn, da dann jeder Punkt unseres Computeruniversums Punkt des Raumes ist. Versuchen wir also, einen Zylinder darzustellen. Legen wir die Grundfläche des Zylinders auf die Ebene, die durch r2 und r3 aufgespannt wird. Da die Grundfläche eines Zylinders ein Kreis ist, können wir die Einschränkung dafür bereits angeben:

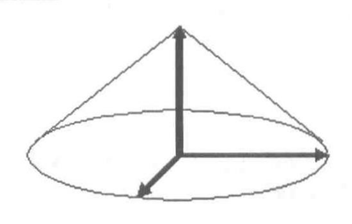
```
u2 + v2 = 1
```

Diesmal darf es nicht kleiner/gleich heißen, da wir eine Röhre haben wollen und keinen ausgefüllten Zylinder. Da unser Zylinder aber nicht unendlich lang sein soll, schränken wir w auf [0,1] ein, so daß wir insgesamt erhalten:

```
u2 + v2 = 1 und 0<=w<=1.
```







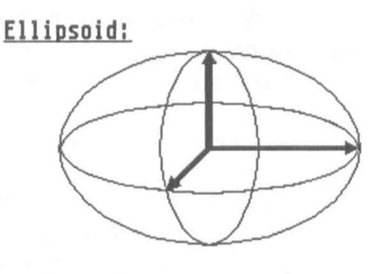


Abbildung 3.8

Fertig ist der Zylinder! Ganz so einfach, wie wir ihn hier definiert haben, läßt er sich später im Programm nicht handhaben, aber es geht. Genau wie bei der Kreisfläche können wir eine weitere Einschränkung angeben, nämlich ein Winkelintervall, so daß wir einen Zylinderausschnitt erhalten. Da das Vorgehen identisch zur Kreisfläche ist, schenken wir uns weitere Erklärungen und wenden uns dem Kegel zu.

Die Grundfläche des Kegels ist ebenfalls ein Kreis, so daß wir einen Teil der Einschränkung bereits zur Genüge kennen. Doch wie erreichen wir, daß der Kegel zu einer Seite hin spitz zusammenläuft? Sehen wir uns einen Längsschnitt eines Kegels an, so erinnert uns dieser an ein Dreieck. Die Einschränkung für ein Dreieck ist ja bekanntlich x+y=1. y ist in unserem Fall w, also der Parameter des Vektors, der zur Spitze des Kegels zeigt. x ist der Vektor, der auf einen Punkt der Grundfläche zeigt: x=SQR(u2 + v2). Während wir das Wurzelziehen bisher vergessen konnten (da der Radius Eins war und SQR(1)=1), müssen wir jetzt daran denken, da das Ergebnis sonst nicht unbedingt einem Kegel ähnlich sieht. Insgesamt erhalten wir:

```
SQR(u2 + v2) + w = 1.
```

Wie bei der Kreisfläche und beim Zylinder können wir auch hier einen Kegelausschnitt erzeugen, indem wir ein Winkelintervall festlegen oder einen Kegelstumpf, indem wir w einschränken.

Den Abschluß unserer Grundobjekte bildet das Ellipsoid. Zwar gäbe es noch mindestens einen weiteren interessanten Grundkörper, doch werden wir diesen hier nicht behandeln, sondern nur in den Ausblicken erwähnen. Momentan stellt sich uns die Frage, ob wir ein Ellipsoid brauchen, da wir ja schon eine Kugel haben. Anders als die Kugel kann das Ellipsoid aber einen beliebigen (ellipsenförmigen) Querschnitt haben und nicht nur einen kreisrunden. Die Grundfläche bildet wieder ein Kreis, aber auch der Längsschnitt ist ein Kreis. Die Einschränkung erhalten wir wieder auf ähnlich Art und Weise wie beim Kegel, nur daß diesmal ein anderes Ergebnis herauskommt:

```
u2 + v2 + w2 = 1.
```

Wählen Sie r2, r3 und r4 gleich groß, so erhalten Sie eine Kugel. Und nun zum zweiten Vorteil des Ellipsoids: Wir können wieder fleißig einschränken, so daß wir einen Ellipsoidausschnitt erhalten. Diesen können Sie sich als Stück einer Apfelsinenschale vorstellen oder als Melonenscheibe, freilich ohne Inhalt. Ebenso ist es möglich, einen der Parameter zusätzlich auf ein Intervall einzuschränken; über die möglichen Ergebnisse können Sie selbst mal nachdenken. Allerdings möchte ich Ihnen hier den Mund nicht allzu wäßrig machen, da in den Routinen, die wir später entwickeln werden, Kegel- und Ellipsoidausschnitte noch nicht enthalten sind. Diese sind sozusagen als Hausaufgabe für Sie gedacht.

3.3.1 Datenstruktur

Zum Schluß müssen wir für die Parameter unserer Objekte noch eine Datenstruktur festlegen. Da wir den Algorithmus in BASIC definieren werden, empfiehlt sich ein mehrdimensionales Feld, welches wir einfach K() nennen, K wie Körper. Für jedes Element von K müssen wir bis zu vier Vektoren abspeichern können, dazu kommen noch Informationen über den Typus des Körpers, dessen Materialkonstanten und die Einschränkungen. Mit K(MaxAnzahl,5,2) sollten wir auskommen. MaxAnzahl ist die maximale Anzahl von Körpern, aus denen unsere Computerwelt zusammengesetzt ist. Dabei sind den Elementen aus K folgende Parameter zugeordnet:

K(n,0,0): Typ des n-ten Körpers. Dieses Element legt fest, ob es sich bei dem n-ten Körper um eine Ebene, eine Kugel oder etwas anderes handelt. Folgende Werte entsprechen dabei folgenden Grundobjekten:

0: Ebene, unendlich.
1: Dreieck.
2: Parallelogramm.
3: Kreisfläche.
4: Kreisausschnitt.
5: Kreisbogen.

10: Kugel.

20: Zylinder.
21: Zylinderausschnitt.
22: Kegel.
24: Ellipsoid.

Wie Sie sehen, sind nicht alle oben erwähnten Einschränkungen schon in die Datenstruktur eingebaut. Dies können Sie nach Lust und Laune nachträglich vornehmen. Während K(n,0,1) für zukünftige Erweiterungen reserviert ist, enthält K(n,0,2) den Index des Materials, aus dem dieses Objekt besteht. Statt alle Materialkonstanten auch im Feld K unterzubringen, sind diese in ein eigenes Feld namens Mat ausgelagert. K(n,0,2) gibt nun die Nummer des Elementes von Mat an, das die Materialkonstanten für das n-te Objekt enthält. Dies hat den Vorteil, daß man auf einfache Weise mehreren Objekten die gleichen Materialkonstanten zuordnen kann. Auch lassen sich so leichter Änderungen an den Materialkonstanten vornehmen, die für mehrere Objekte definiert sind.

K(n,1,.) enthält immer den Vektor r1. Dabei ist K(n,1,0) die X-Komponente, K(n,1,1) die Y-Komponente und K(n,1,2) die Z-Komponente des Vektors, was auch allgemein für die anderen Vektoren gilt. Je nachdem, wieviele Vektoren für das jeweilige Objekt benötigt werden, enthalten K(n,2,.) bis K(n,4,.) die Vektoren r2 bis r4. Im Falle der Kugel enthält K(n,2,0) den Radius.

Hinweis: Der Punkt in der Klammer steht für einen beliebigen erlaubten Wert. K(n,1,.) meint also K(n,1,0), K(n,1,1) und K(n,1,2).

Die Einschränkungen überschneiden sich teilweise mit dem Vektor r4, was aber nicht weiter stört, da genau die Einschränkung, die dort liegt, nicht für Zylinder, Kegel und Ellipsoid Anwendung findet. Für den Kreisbogen enthält K(n,4,0) die untere und K(n,4,1) die obere Grenze für SQR(u2+v2), wobei die Werte zwischen Null und Eins liegen müssen. Für Kreisausschnitt, -bogen und Zylinderausschnitt enthält K(n,5,0) den Start- und K(n,5,1) den Endwinkel in Bogenmaß. Zusammengefaßt ergibt sich folgende Aufteilung:

```
K(n,0,0) = Typ.
K(n,0,1) = reserviert.
K(n,0,2) = Index für Materialkonstanten.

K(n,1,.) = r1.

K(n,2,0) = Radius der Kugel oder
K(n,2,.) = r2.

K(n,3,.) = r3.

K(n,4,.) = r4 oder falls Kreisbogen:
K(n,4,0) = untere Grenze (innerer Radius)
K(n,4,1) = obere Grenze (äußerer Radius).
```

Falls Kreisausschnitt, -bogen oder Zylinderausschnitt:

```
K(n,5,0) = Startwinkel,
K(n,5,1) = Endwinkel.
```

Die Datenstruktur für die Materialkonstanten ist da einfacher. Mit Mat(AnzahlMat,6) ist das Feld dimensioniert. Zuerst ist die Farbe des Materials festgelegt: Mat(m,0) enthält den Rot-Anteil, Mat(m,1) den Grün-Anteil und Mat(m,2) den Blau-Anteil. Da es in unserer Computerwelt kein Streulicht gibt und somit alle Schatten tiefschwarz wären, definieren wir für jedes Material eine Schattenhelligkeit, welche angibt, wie hell das Material im Schatten ist. Hier sollten Sie bei Definition von Materialien keinen zu großen Wert eingeben, da man sonst Schatten und beleuchtete Stellen nur noch schwer voneinander trennen kann.

Mat(m,4) enthält den Spiegelungsfaktor. Ist dieser Null, so ist die Oberfläche des Materials völlig stumpf, ist er Eins, so entspricht dies einem auf Hochglanz poliertem Spiegel. Mat(m,5) enthält den Transparenzfaktor, den wir aber noch nicht benutzen werden, und Mat(m,6) ist reserviert für Erweiterungen. Die Datenstruktur für Mat im Überblick:

```
Mat(m,0) = Rot-Anteil der Farbe.
Mat(m,1) = Grün-Anteil der Farbe.
Mat(m,2) = Blau-Anteil der Farbe.

Mat(m,3) = Schattenhelligkeit.

Mat(m,4) = Spiegelungsfaktor.
Mat(m,5) = Transparenzfaktor.

Mat(m,6) = reserviert.
```

Alle Werte müssen im übrigen zwischen Null und Eins liegen.

Wenn Sie später mal wissen wollen, welchen Spiegelungsfaktor der Körper mit der Nummer 47 hat, so lassen Sie sich erst den Materialindex ausgeben:

```
PRINT K(47,0,2)
```

Nehmen wir an, Sie erhalten den Wert 12. Dann können Sie sich mit

```
PRINT Mat(12,4)
```

den Spiegelungsfaktor ausgeben lassen. Schneller geht es, wenn Sie direkt schreiben:

```
PRINT Mat(K(47,0,2),4)
```

Das sieht zwar wüster aus, spart aber einen Zwischenschritt und ist deswegen auch in den Routinen verwendet worden.

3.4 Simulation der Wahrnehmung

Nachdem wir uns in unserem Computer eine künstliche Wirklichkeit zusammengebastelt haben, geht es nun darum, den Computer dazu zu bringen, diese auf dem Bildschirm anzuzeigen. Als erstes gilt es festzulegen, wohin wir sehen wollen.

In unserem Simulationsmodell haben wir Sehstrahlen von jedem Punkt unseres Bildschirms aus losgeschickt, und zwar in Richtung der Linse. In unserem Algorithmus werden wir anders vorgehen. Wir werden von einem festen Punkt, dem Projektionspunkt P, Sehstrahlen in Richtung des Bildschirms losschicken und zwar für jeden Bildschirmpixel genau einen Sehstrahl. Nun müssen wir noch die Position des Bildschirms festlegen. Zu diesem Zweck definieren wir den Hauptpunkt H, welcher exakt in der Mitte des Bildschirms liegt. Der Vektor (P-H) ist dabei Normalenvektor auf unseren Bildschirm, das heißt, das sich der Projektionspunkt senkrecht über der Bildschirmmitte befindet. Den Abstand zwischen P und H nennen wir dph.

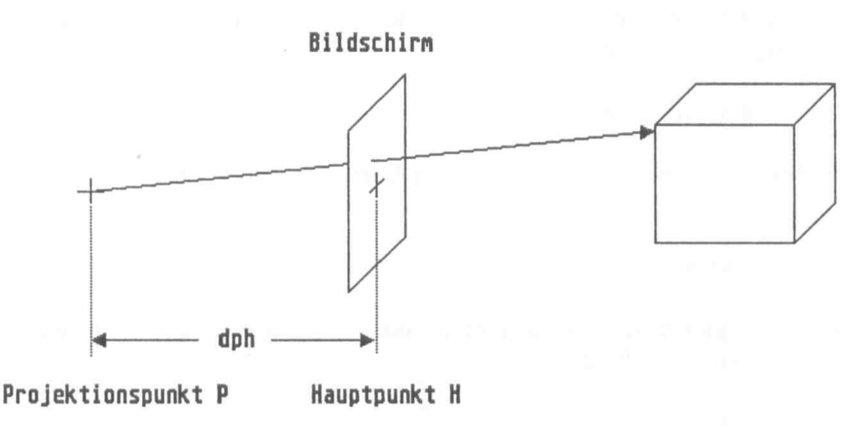


Abbildung 3.9

Da wir für ein Bild ohnehin alle Bildpunkte ausrechnen müssen, ist der grobe Algorithmus denkbar einfach: Für jeden Bildpunkt, den wir berechnen wollen, bestimmen wir die dreidimensionalen Koordinaten. Dann schicken wir vom Projektionspunkt einen

Sehstrahl in Richtung dieses Punktes los und stellen fest, ob wir etwas sehen, ob wir nichts sehen und was wir sehen, also insbesondere die Farbe des Punktes. Ist diese bestimmt, so geben wir den Punkt auf dem Bildschirm aus. Als Unterprogramm in BASIC sieht das Ganze wie folgt aus:

```
Schattiere:

  'Initialisierungen:

  Status$ = " Status: Schattieren"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  Status$ = " Status: Init"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
  GOSUB InitSchattiere

  Status$ = " Status: InitMinMax"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
  GOSUB InitMinMax

  Status$ = " Status: InitMinMaxLq"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
  GOSUB InitMinmaxLq

  CALL Scron

  Status$ = " Status: Schattieren"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  XAOff = (BoxW% AND 1)*.5
  YAOff = (BoxH% AND 1)*.5

  ' Offsets, um richtige Punktgröße zu garantieren:

  IF XAOff = .5 THEN
    XBOff = .5
  ELSE
    XBOff = 1
  END IF
  IF YAOff = .5 THEN
    YBOff = .5
```

```
ELSE
  YBOff = 1
END IF

'Nun folgt das eigentliche Berechnen:

Koerper% = 0

FOR yb% = YStart%+BoxH%/2 TO Yend%+BoxH%/2 STEP BoxH%
  FOR xb% = XStart%+BoxW%/2 TO XEnd%+BoxW%/2 STEP BoxW%

    CALL Reprojektion(Rx,Ry,Rz,0!,FN Xresc(xb%),FN Yresc(yb%))

    Rx = Rx-Px
    Ry = Ry-Py
    Rz = Rz-Pz

    StackPtr%  = -1

    Px1 = Px
    Py1 = Py
    Pz1 = Pz

    Original! = True
    GOSUB Berechnepunkt

    Px = Px1
    Py = Py1
    Pz = Pz1

    IF Stack(0,0)> = 0 THEN  'Falls kein Schnittpunkt  = > -1
      Hell.r = Stack(0,0)*Qh.r
      Hell.g = Stack(0,1)*Qh.g
      Hell.b = Stack(0,2)*Qh.b
      CALL OSSetPoint&(CLNG(xb%-BoxW%/2+XAOff),
                       CLNG(yb%-BoxH%/2+YAOff),
                       CLNG(xb%+BoxW%/2-XBOff),
                       CLNG(yb%+BoxH%/2-YBOff),
                       CLNG(1024*Hell.r),
                       CLNG(1024*Hell.g),
                       CLNG(1024*Hell.b))
    END IF
  NEXT xb%

  IF INKEY$ <> "" THEN
    IF yb% < Yend%+BoxH%/2 THEN
```

```
          CALL Scroff
          a$ = "Nächste Zeile:"+STR$(yb%+BoxH%)
          CALL Ausgabe(a$,130,False)
          CALL DialogBox("Weiter",0,True,x1a%,y1a%,x2a%,y2a%,False)
          CALL DialogBox("Stop",2,False,x1b%,y1b%,x2b%,y2b%,False)

          CALL DoDialog(n%,0,x1a%,y1a%,x2a%,y2a%,-1,-1,
          -1,-1,x1b%,y1b%,x2b%,y2b%)

          CLS
          IF n% = 0 THEN

            CALL Scron
           ELSE
            GOTO SchattEnde
          END IF
        END IF
      END IF
    NEXT yb%

    SchattEnde:

    CALL Scroff
    CLS
    GOSUB MakeMenu
  RETURN
```

Zuerst finden einige Initialisierungen statt, die uns vorerst nicht interessieren sollen; wir werden darauf später noch eingehen. Der wesentliche Teil beginnt mit den beiden FOR-TO-Schleifen, die sämtliche Bildschirmpunkte abfragen. BoxW% und BoxH% sind Breite und Höhe des Bildschirmpunktes in Pixel, werden also normalerweise beide gleich Eins sein. Als erstes innerhalb der Schleife werden dann aus den zweidimensionalen Bildschirmkoordinaten die dreidimensionalen Koordinaten des aktuellen Bildschirmpunktes berechnet.

Reprojektion arbeitet dabei als Umkehrfunktion zu Projektion, welche dreidimensionale Koordinaten auf den Bildschirm projiziert. Die beiden Funktionen Xresc und Yresc dienen dazu, eventuell eingestellte Vergrößerungen zu berücksichtigen. Als nächstes wird der Vektor bestimmt, der von P aus in Richtung

des Bildschirmpunktes zeigt (Rx,Ry,Rz). Dann wird P gesichert und die Funktion BerechnePunkt aufgerufen, die die Farbe des Bildschirmpunktes berechnet. Das Ergebnis der Berechnung wird in dem nullten Element von Stack() abgelegt und zwar der Rotanteil der Farbe in Stack(0,0), der Grünanteil in Stack(0,1) und der Blauanteil in Stack(0,2). Wofür wir den Stack brauchen, ahnen Sie sicher bereits, nämlich für die Spiegelung.

Wenn Stack(0,0) negativ ist, so ist in diesem Bildschirmpunkt nichts zu sehen, und somit lassen wir dort den Hintergrund so wie er war. Andernfalls berücksichtigen wir die Farbe der Lichtquelle (Qh.r = Rotanteil, Qh.g = Grünanteil und Qh.b = Blauanteil) und geben den Punkt mit der Routine OSSetPoint& aus, die ein Problem für sich darstellt. Diese Routine zeichnet ein Rechteck auf den Bildschirm, wobei sie versucht, der gewünschten Farbe so nahe wie möglich zu kommen, doch auch dazu später mehr.

Nachdem eine Zeile berechnet ist, wird abgefragt, ob der Benutzer zwischenzeitlich eine Taste gedrückt hat, und wenn ja, wird ihm angeboten, die Berechnung abzubrechen. Die Scron/ Scroff-Anweisungen haben nur den Zweck, den Grafikbildschirm nach vorne oder hinten zu legen. Wie der Benutzerdialog funktioniert, können Sie an anderer Stelle in diesem Buch lesen, so daß wir uns nun der Frage aller Fragen zuwenden können.

3.4.1 Sehen wir was?

Als erstes gilt es,die Frage zu beantworten, ob wir überhaupt etwas sehen. Erst wenn wir das geschafft haben, können wir uns der Frage zuwenden, was wir denn nun sehen.

Natürlich haben wir die Grundobjekte nicht umsonst durch mathematische Gleichungen definiert. Diese haben nämlich den Vorteil, daß man den Schnittpunkt dieser Objekte mit einer Geraden relativ einfach berechnen kann. Die Gerade, um die es hier geht, ist unser Sehstrahl. Dieser beginnt im

Projektionspunkt P und läuft durch den aktuellen Bildschirmpunkt. Eine Geradengleichung kann man als Punkt-Richtungs-Form aufschreiben:

```
r = r1 + l*r2
```

wobei r, r1 und r2 Vektoren sind und l ein Skalar ist (siehe auch Abbildung 3.3). Der Punkt r1, durch den die Gerade geht, entspricht unserem Projektionspunkt. r2 gibt die Richtung der Geraden an, was unserem Vektor R (Rx,Ry,Rz) entspricht, den wir oben berechnet haben. Um den Schnittpunkt mit einer Ebene zu bestimmen, setzen wir die Geraden- und die Ebenengleichung gleich, so daß wir die Parameter l, u und v ausrechnen können. Den Schnittpunkt erhalten wir, indem wir l in die Geradengleichung einsetzen und diese ausrechnen. Als Unterprogramm sieht die Schnittpunktberechnung folgender-maßen aus:

```
SUB SchnittpunktEbene(n%,Px,Py,Pz,Rx,Ry,Rz,l) STATIC
SHARED Hilf(),K()
  '  => l = Geradenparameter

  'Zuerst Nennerdeterminante berechnen:

  d = Ry*Hilf(n%,4)-Rx*Hilf(n%,3)-Rz*Hilf(n%,5)
  IF d<>0 THEN
    l = ((Px-K(n%,1,0))*Hilf(n%,3)-(Py-K(n%,1,1))*Hilf(n%,4)
      +(Pz-K(n%,1,2))*Hilf(n%,5))/d
  ELSE

    ' Es existiert kein Schnittpunkt:

    l = -1
  END IF
END SUB
```

Übergeben wird der feste Punkt der Gerade (Px,Py,Pz) und der Richtungsvektor (Rx,Ry,Rz). Außerdem wird mit n% festgelegt, mit welcher Ebene der Schnittpunkt berechnet werden soll. Zurückgegeben wird von der Subroutine der Geradenparameter l. Existiert kein Schnittpunkt, so wird l = -1 zurückgegeben. Nun

werden Sie sagen, daß l durchaus auch negativ sein kann. Dies stimmt zwar, ab dann läge der Schnittpunkt "hinter" uns. Uns interessieren aber nur die Sachen, die wir vor uns sehen.

Wie man Gleichungssysteme löst, nämlich über Determinanten, können Sie gegebenenfalls in den mathematischen Grundlagen nachlesen. Zuerst wird die Nennerdeterminante d berechnet. Diese sieht wie folgt aus:

Gerade: $r = r_P + l*r_R$, Ebene: $r = r_1 + u*r_2 + v*r_3$

Schnittpunkt: $l*r_R - u*r_2 - v*r_3 = (r_1 - r_P)$

$$D = \begin{vmatrix} Rx & K(n,2,0) & K(n,3,0) \\ Ry & K(n,2,1) & K(n,3,1) \\ Rz & K(n,2,2) & K(n,3,2) \end{vmatrix} \quad \text{Nennerdeterminante}$$

$$D_L = \begin{vmatrix} K(n,1,0)-Px & K(n,2,0) & K(n,3,0) \\ K(n,1,1)-Py & K(n,2,1) & K(n,3,1) \\ K(n,1,2)-Pz & K(n,2,2) & K(n,3,2) \end{vmatrix} \qquad \boxed{l = \frac{D_L}{D}}$$

$$D_U = \begin{vmatrix} Rx & K(n,1,0)-Px & K(n,3,0) \\ Ry & K(n,1,1)-Py & K(n,3,1) \\ Rz & K(n,1,2)-Pz & K(n,3,2) \end{vmatrix} \qquad \boxed{u = \frac{D_U}{D}}$$

$$D_V = \begin{vmatrix} Rx & K(n,2,0) & K(n,1,0)-Px \\ Ry & K(n,2,1) & K(n,1,1)-Py \\ Rz & K(n,2,2) & K(n,1,2)-Pz \end{vmatrix} \qquad \boxed{v = \frac{D_V}{D}}$$

Abbildung 3.10

Da von dieser Determinante nur eine Zeile unbekannt ist, nämlich die, in die der Richtungsvektor der Geraden eingesetzt wird, lösen wir die Determinante auch nach dieser Zeile auf. Die 2x2-Unterdeterminanten haben alle einen Wert, der sich

wärend der Berechnung eines ganzen Bildes nicht ändert. Daher rechnen wir deren Werte vorher aus und speichern sie im Feld Hilf() ab. Es seien:

```
Hilf(n%,3) = K(n%,2,1)*K(n%,3,2)-K(n%,3,1)*K(n%,2,2)
Hilf(n%,4) = K(n%,2,0)*K(n%,3,2)-K(n%,3,0)*K(n%,2,2)
Hilf(n%,5) = K(n%,2,0)*K(n%,3,1)-K(n%,3,0)*K(n%,2,1)
```

Ist die Nennerdeterminante gleich Null, so existiert kein Schnittpunkt, weil Gerade und Ebene parallel liegen. Sonst wird l direkt ausgerechnet und an das aufrufende Programm übergeben.

Bevor wir uns gleich der Schnittpunktberechnung mit anderen Grundobjekten zuwenden, noch ein paar Worte zum generellen Vorgehen: Wir bestimmen für jedes Objekt, aus dem unsere Computerwelt besteht, den Schnittpunkt mit dem Sehstrahl. Dabei merken wir uns stets die Nummer des Objektes, das uns bisher am nächsten war; wir suchen also ein Objekt mit minimalem l. Haben wir dies gefunden, haben wir auch die Nummer des Objektes, das in dem momentanen Bildschirmpunkt zu sehen ist.

Doch nun zu den Schnittpunkten mit weiteren Grundobjekten. Für das Dreieck sieht das Unterprogramm identisch aus, nur daß zusätzlich die Parameter u und v berechnet werden müssen und dann getestet wird, ob diese der Einschränkung für Dreiecke genügen:

```
SUB SchnittpunktDreieck(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b) STATIC
SHARED Hilf(),K()
  ' = > l = Geradenparameter, a,b = Flächenparameter

  'Zuerst wieder die Nennerdeterminante:

  d = Ry*Hilf(n%,4)-Rx*Hilf(n%,3)-Rz*Hilf(n%,5)
  IF d<>0 THEN

    l = ((Px-K(n%,1,0))*Hilf(n%,3)-(Py-K(n%,1,1))*Hilf(n%,4)
      +(Pz-K(n%,1,2))*Hilf(n%,5))/d

    IF l>0 THEN

      'Nun die beiden Parameter u und v bestimmen:
      'Lassen Sie sich nicht dadurch verwirren,
```

```
      'daß diese hier a und b heißen. Dies tut nichts zur Sache!

      a = FN Det(Px-K(n%,1,0),K(n%,3,0),-Rx,Py-
K(n%,1,1),K(n%,3,1),
               -Ry,Pz-K(n%,1,2),K(n%,3,2),-Rz)/d
      b = FN Det(K(n%,2,0),Px-K(n%,1,0),-Rx,K(n%,2,1),Py-
K(n%,1,1),
               -Ry,K(n%,2,2),Pz-K(n%,1,2),-Rz)/d

      'Nun die Einschränkung: Ungekehrt formuliert

      IF a<0 OR b<0 OR a>1 OR b>1 OR a+b>1 THEN
        l = -1
      END IF

    END IF
  ELSE
    l = -1
  END IF
END SUB
```

Die Funktion Det berechnet nur den Wert einer Determinanten, ansonsten dürfte Ihnen alles andere bekannt sein. Analog laufen die Schnittpunktberechnungen für Parallelogramm, Kreis, Kreisausschnitt und Kreisbogen ab, nur daß die Einschränkung eine andere ist. Die Namen der Unterprogramme sind in Klammern dahinter angegeben:

```
Parallelogramm (SchnittpunktViereck):
IF a<0 OR b<0 OR a>1 OR b>1 THEN
  l = -1
END IF

Kreisfäche (SchnittpunktKreis):
IF a*a+b*b>1 THEN
  l = -1
END IF

Kreisausschnitt (SchnittpunktKreisSektor):
IF a*a+b*b< = 1 THEN
  CALL WinkelIntervall(l,n%,a,b)
ELSE
  l = -1
END IF
```

```
Kreisbogen (SchnittpunktKreisRing):
d = SQR(a*a+b*b)
IF (d> = K(n%,4,0)) AND (d< = K(n%,4,1)) THEN
  CALL WinkelIntervall(l,n%,a,b)
ELSE
  l = -1
END IF
```

WinkelIntervall testet dabei, ob der Schnittpunkt innerhalb des definierten Winkelintervalls liegt:

```
SUB WinkelIntervall(l,n%,a,b) STATIC
SHARED Pi,Pm2,Pd2,K()

  'Winkel ausrechnen:

  IF a = 0 THEN
    d = Pd2*SGN(b)
  ELSE
    d = ATN(b/a)
    IF a<0 THEN
      d =  d+Pi
    END IF
  END IF
  IF d<0 THEN
    d =  d+Pm2
  END IF
```

Und nun abfragen, ob dieser im definierten Intervall liegt:

```
  IF d<K(n%,5,0) OR d>K(n%,5,1) THEN
    l = -1
  END IF
END SUB
```

Pi ist dabei die Kreiszahl 3.1415927, Pm2 = 2*Pi und Pd2 = Pi/2. Hier sehen Sie auch im Detail, wie man den Winkel berechnen kann.

Bei der Kugel setzt man die Geradengleichung in die Kugelgleichung ein und rechnet diese dann aus. Wir erhalten ein quadratisches Gleichungssystem, das wir mit dem Standardverfahren auflösen. Einziges Problem: Wir erhalten maximal zwei Schnittpunkte und nicht mehr nur einen einzigen. Für beide müssen wir den Parameter l berechnen. Dann wählen wir den aus, der uns am nächsten liegt.

```
SUB SchnittpunktKugel(n%,Px,Py,Pz,Rx,Ry,Rz,l) STATIC
SHARED Hilf(),K(),Schwelle
  ' = > l = Geradenparameter
  d = Rx*Rx+Ry*Ry+Rz*Rz
  p = (Rx*(Px-K(n%,1,0))+Ry*(Py-K(n%,1,1))+Rz*(Pz-K(n%,1,2)))/d
  q = ((Px-K(n%,1,0))^2+(Py-K(n%,1,1))^2+(Pz-K(n%,1,2))^2
     -K(n%,2,0)*K(n%,2,0))/d
  d = p*p-q
  IF d> = 0 THEN
    l = -p+SQR(d)          'erste Lösung
    L1 = -p-SQR(d)         'zweite Lösung
    IF (L1<l) AND (L1>Schwelle) THEN

      'L1 ist uns näher als l

      SWAP l,L1
    END IF
  ELSE
    l = -1
  END IF
END SUB
```

Wir testen diesmal nicht, ob L1 größer als Null ist, sondern ob es größer als eine Schwelle ist, die sich durch die Ungenauigkeit des Rechners ergibt und für die einfachgenauen Zahlen des Amiga auf 0.0001 gesetzt wurde. Diese Schwelle wird aber erst beim Schatten und bei der Spiegelung von Interesse sein. Da diese aber mit denselben Routinen arbeiten, muß die Abfrage hier bereits eingebaut sein.

Bis jetzt waren die Routinen zur Schnittpunktberechnung ja noch einigermaßen manierlich. Doch nun kommen wir zum Zy-

linder und seinen Verwandten, wobei wir auf wahre Formelschlachten stoßen werden. Empfindliche Gemüter mögen daher die Augen schließen oder mit dem nächsten Kapitel weitermachen.

Beim Zylinder haben wir in der Zylindergleichung drei und in der Geradengleichung eine Unbekannte, so daß Gleichsetzen alleine nicht ausreicht. Zwar haben wir noch die Einschränkung zur Vefügung, müssen dann aber 4x4-Determinanten ausrechnen. Daher gehen wir anders vor.

Durch die drei Richtungsvektoren r2, r3 und r4 wird ja wieder ein Koordinatensystem aufgespannt. Wenn wir also die Koordinaten von P und R in dieses Koordinatensystems umwandeln (Basistransformation), so können wir auch dort den Schnittpunkt der Geraden mit den Zylinder berechnen. Die Zylindergleichung reduziert sich zu:

```
u^2 + v^2 = 1 und 0< = w< = 1
```

wobei diesmal (u,v,w) der Schnittpunkt mit der Geraden sein soll. Die Geradengleichung, aufgespalten in die drei Koordinatengleichungen, lautet dann:

```
u = Pxt + l*xt
v = Pyt + l*yt
w = Pzt + l*zt
```

wobei (Pxt,Pyt,Pzt) der transformierte Projektionspunkt und (xt,yt,zt) der transformierte Richtungsvektor ist. Setzen wir dies in die Zylindergleichung ein, so erhalten wir:

```
(Pxt + l*xt)^2 + (Pyt + l*yt)^2 = 1
0 < = (Pzt + l*zt) < = 1
```

Hier ist jetzt nur noch l unbekannt und läßt sich nach einigen Umformungen ausrechnen:

```
l^2*(xt^2+yt^2) + l*(2*Pxt*xt+2*Pyt*yt) + (Pxt^2+Pyt^2-1) = 0
```

Das weitere Ausrechnen dieser Gleichung wird auch aus dem Unterprogramm ersichtlich, so daß wir uns das gesonderte Aufschreiben der Lösung ersparen.

```
SUB SchnittpunktZylinder(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b,c,Original!)
STATIC
SHARED Schwelle
  ' = > l = Geradenparameter,
  '     a,b,c = transformierte Schnittpunktskoordinaten

  'Zuerst P und R transformieren:

  CALL
Basistrans(n%,Pxt,Pyt,Pzt,Xt,Yt,Zt,Px,Py,Pz,Rx,Ry,Rz,Original!)

  d = Xt*Xt+Yt*Yt
  IF d<>0 THEN
    p = (Pxt*Xt+Pyt*Yt)/d
    q = (Pxt*Pxt+Pyt*Pyt-1)/d
    d = p*p-q
    IF d> = 0 THEN
      l = -p+SQR(d)
      L1 = -p-SQR(d)
      c = Pzt+l*Zt
      d = Pzt+L1*Zt

      'Zwei Lösungen: l,c und L1,d. Welche ist die richtige?

      IF L1<l AND L1>Schwelle AND d> = 0 AND d< = 1 THEN
        SWAP l,L1
        SWAP c,d
      END IF

      'l und c enthalten die richtigen Lösungsparameter:

      IF c<0 OR c>1 THEN
        l = -1
      ELSE
        a = Pxt+l*Xt
        b = Pyt+l*Yt
      END IF
    ELSE
```

```
      l = -1
    END IF
  ELSE
    l = -1
  END IF
END SUB
```

Auch hier gibt es, wie schon bei der Kugel, zwei Lösungen, die beide berechnet werden müssen und von denen der Schnittpunkt gewählt wird, der uns am nächsten liegt. c und d entsprechen dabei den beiden Lösungen für w und müssen daher zwischen Null und Eins liegen. Existiert der Schnittpunkt, so werden auch a und b, die Lösungen für u und v, ausgerechnet.

Bei der Parameterübergabe ist diesmal noch ein zusätzlicher Parameter dazugekommen: Original!. Dieser wird bei BasisTrans benötigt. BasisTrans rechnet die Vektoren P und R in das Koordinatensystem des Zylinders um, führt also die bereits erwähnte Basistransformation aus:

```
SUB
Basistrans(n%,Pxt,Pyt,Pzt,Xt,Yt,Zt,Px,Py,Pz,Rx,Ry,Rz,Original!)
    STATIC
SHARED Hilf(),K()
  ' Transformiert (px,py,zy)  = > (pxt,pyt,pzt), (rx,ry,rz)  = >
    (xt,yt,zt)

  IF Original! = True THEN
    Pxt = Hilf(n%,10)
    Pyt = Hilf(n%,11)
    Pzt = Hilf(n%,12)
  ELSE
    Pxt = (Px-K(n%,1,0))*Hilf(n%,13)-(Py-K(n%,1,1))*Hilf(n%,14)
       +(Pz-K(n%,1,2))*Hilf(n%,15)
    Pyt = (Px-K(n%,1,0))*Hilf(n%,16)-(Py-K(n%,1,1))*Hilf(n%,17)
       +(Pz-K(n%,1,2))*Hilf(n%,18)
    Pzt = (Px-K(n%,1,0))*Hilf(n%,19)-(Py-K(n%,1,1))*Hilf(n%,20)
       +(Pz-K(n%,1,2))*Hilf(n%,21)
  END IF
```

```
    Xt = (Px+Rx-K(n%,1,0))*Hilf(n%,13)-(Py+Ry-K(n%,1,1))*Hilf(n%,14)
      +(Pz+Rz-K(n%,1,2))*Hilf(n%,15)-Pxt
    Yt = (Px+Rx-K(n%,1,0))*Hilf(n%,16)-(Py+Ry-K(n%,1,1))*Hilf(n%,17)
      +(Pz+Rz-K(n%,1,2))*Hilf(n%,18)-Pyt
    Zt = (Px+Rx-K(n%,1,0))*Hilf(n%,19)-(Py+Ry-K(n%,1,1))*Hilf(n%,20)
      +(Pz+Rz-K(n%,1,2))*Hilf(n%,21)-Pzt
END SUB
```

Das Schöne daran ist nun, daß sich die transformierten Koordinaten des Projektionspunkts während der gesamten Berechnung nicht ändern. Wir können diese also zuvor ausrechnen und in unserem Feld Hilf abspeichern. Da die Schnittpunktroutinen aber auch für Spiegelungs- und Schattenberechnung verwendet werden, wo statt des Projektionspunktes ein anderer Punkt in (Px,Py,Pz) übergeben wird, müssen wir eine diesbezügliche Abfrage einbauen, was mit dem IF-THEN-ELSE-Konstrukt geschieht. Wenn Original! True ist, so enthalten (Px,Py,Pz) die Koordinaten des ursprünglichen Projektionspunktes.

Die Basistransformation ist wieder die Lösung eines Gleichungssystems mit drei Unbekannten. Dabei wird der zu transformierende Punkt einfach in die Körpergleichung eingesetzt, was für P wie folgt aussieht:

```
P = r1 + u*r2 + v*r3 + w*r4
```

oder in die Koordinatengleichungen aufgelöst:

```
Px = r1x + u*r2x + v*r3x + w*r4x
Py = r1y + u*r2y + v*r3y + w*r4y
Pz = r1z + u*r2z + v*r3z + w*r4z
```

Das Ganze muß nun aufgelöst und die Parameter u, v und w müssen ausgerechnet werden. Zugute kommt uns dabei, daß die Nennerdeterminante diesmal erfreulicherweise gleich Eins ist, so daß wir sie nicht extra berechnen müssen. Bei der Ausrechnung der Determinante können wir uns wieder zunutze machen, daß dort nur eine Spalte jeweils unbekannt ist, während die beiden anderen konstant sind, so daß wir diese auch vorher berechnen und in Hilf abspeichern können, wie wir es schon bei der Ebene getan haben.

Damit wäre auch der Schnittpunkt einer Geraden mit unserem Zylinder berechnet. Für den Zylinderausschnitt müssen wir nur für die beiden Parameter a und b die Abfrage bezüglich des Winkelintervalls einbauen:

```
(SchnittpunktZylSegm)
l = -p+SQR(d)
L1 = -p-SQR(d)
c = Pzt+l*Zt
C1 = Pzt+L1*Zt

'Zuerst für Lösung c,l WinkelIntervall testen:

IF c> = 0 AND c< = 1 THEN
  CALL WinkelIntervall(l,n%,Pxt+l*Xt,Pyt+l*Yt)
ELSE
  l = -1
END IF

'Jetzt für die zweite Lösung:

IF C1> = 0 AND C1< = 1 THEN
  CALL WinkelIntervall(L1,n%,Pxt+L1*Xt,Pyt+L1*Yt)
ELSE
  L1 = -1
END IF

'Wenn die zweite Lösung besser ist: Vertauschen

IF (l<-.5) OR (L1<l AND L1>Schwelle) THEN
  SWAP l,L1
  SWAP c,C1
END IF

'Richtige Lösung in c,l

a = Pxt+l*Xt
b = Pyt+l*Yt
```

Auch die Unterprogramme für Kegel und Ellipsoid unterscheiden sich nicht besonders von dem für den Zylinder. Das Verfahren ist identisch, nur sehen die Gleichungen etwas anders aus, da auch die Einschränkungen eine andere Form haben:

```
Kegel (SchnittpunktKegel):
d = Xt*Xt+Yt*Yt-Zt*Zt
IF d<>0 THEN
  p = (Pxt*Xt+Pyt*Yt+Zt*(1-Pzt))/d
  q = (Pxt*Pxt+Pyt*Pyt-(1-Pzt)^2)/d
  d = p*p-q

  ...

END IF

Ellipsoid (SchnittpunktEllipsoid):
d = Xt*Xt+Yt*Yt+Zt*Zt
IF d<>0 THEN
  p = (Pxt*Xt+Pyt*Yt+Pzt*Zt)/d
  q = (Pxt*Pxt+Pyt*Pyt+Pzt*Pzt-1)/d
  d = p*p-q

  ...

END IF
```

Diese beiden Programmstücke finden Sie in den entsprechenden Unterprogrammen jeweils direkt hinter dem BasisTrans-Aufruf.

Wir können nun ganz einfach feststellen, welchen Körper wir im Bildpunkt bx,by sehen, indem wir (bx,by) in dreidimensionale Koordinaten umrechnen und die Gerade bestimmen, die durch P und diesen Punkt R geht. Dann berechnen wir sämtliche Schnittpunkte aller Körper mit dieser Geraden und merken uns jeweils denjenigen, der uns am nächsten ist. Das entsprechende Unterprogramm nennen wir WelcherKoerper, weil es feststellt, welcher Körper von der Geraden geschnitten wird:

```
SUB WelcherKoerper (Kp%,Px,Py,Pz,Rx,Ry,Rz,Original!,Schatten!) STATIC
SHARED True,False,minmax%(),AnzahlK,minmaxlq(),Hilf(),K(),xb%,yb%,
       Sx,Sy,Sz,Koerper%,Ac,Bc,Cc,la,Schwelle

  ' = > Koerper% = Nr des Körpers unter Koordinate xb%,yb%,
  '    S(sx,sy,sz) = Schnittpunkt, la = Geradenabschnitt
  ' = > falls Körper = Zylinder, Kegel, Ellipsoid:
  '      (ac,bc,cc) = transformierte Schnittpunktskoordinaten
```

```
la = -1
Koerper% = 0   'Noch keinen gefunden!

FOR n% = 1 TO AnzahlK

  IF K(n%,0,0) = 0 THEN
    CALL SchnittpunktEbene(n%,Px,Py,Pz,Rx,Ry,Rz,l)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 1 THEN
    CALL SchnittpunktDreieck(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 2 THEN
    CALL SchnittpunktViereck(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 3 THEN
    CALL SchnittpunktKreis(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 4 THEN
    CALL SchnittpunktKreisSektor(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 5 THEN
    CALL SchnittpunktKreisRing(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 10 THEN
    CALL SchnittpunktKugel(n%,Px,Py,Pz,Rx,Ry,Rz,l)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 20 THEN
    CALL SchnittpunktZylinder(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b,c,Original!)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 21 THEN
    CALL SchnittpunktZylinderSegm(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b,c,
    Original!)
    GOTO Wkok
  END IF
  IF K(n%,0,0) = 22 THEN
    CALL SchnittpunktKegel(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b,c,Original!)
    GOTO Wkok
```

```
    END IF
    IF K(n%,0,0) = 24 THEN
      CALL SchnittpunktEllipsoid(n%,Px,Py,Pz,Rx,Ry,Rz,l,a,b,c,Original!)
    END IF
    Wkok:
    ' Work OK!

    IF (l>Schwelle) AND (la< = 0 OR l<la) AND (n%<>Kp% OR K(n%,0,0)>
    = 10)
    THEN
      la = l
      Koerper% = n%
      IF K(n%,0,0)> = 20 AND Kp% = 0 THEN
        Ac = a
        Bc = b
        Cc = c
      END IF
    END IF
    Nxtk:
    ' Nächster Körper
  NEXT n%

  IF Koerper%>0 AND Kp% = 0 THEN
    Sx = Px+la*Rx
    Sy = Py+la*Ry
    Sz = Pz+la*Rz
  END IF
END SUB
```

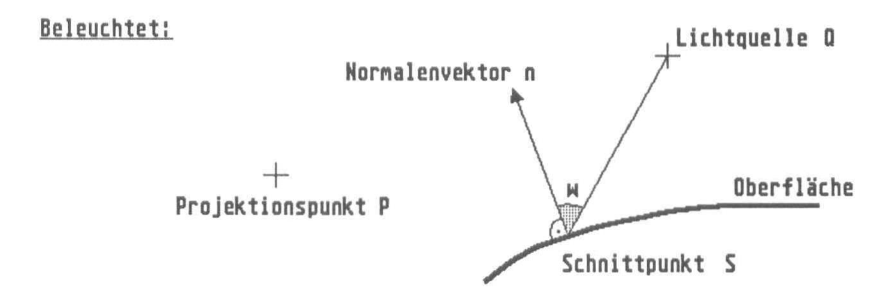
Beim Feststellen, welcher Körper an der momentanen Bildschirmposition sichtbar ist, ist Original! auf True gesetzt. False wird es erst, wenn es um die Spiegelung geht. Ebenso ist Schatten! auf False gesetzt, was sich erst ändert, wenn der Schatten berechnet wird. Außerdem entspricht die hier abgedruckte Routine nicht der, die auf der Diskette zu finden ist. Diese enthält nämlich noch die Optimierungen, auf die wir später noch zu sprechen kommen und die Sie hier nur verwirren würden.

In (Px,Py,Pz) wird der feste Punkt der (Sehstrahl-)Geraden übergeben und in (Rx,Ry,Rz) der Richtungsvektor. Als Ergebnis erhalten Sie in Koerper% die Nummer des Körpers zurück, der (Px,Py,Pz) am nächsten liegt. Kp% wird beim Schatten gebraucht und daher auch erst dort erklärt.

3.4.2 Was sehen wir?

Natürlich reicht es nicht aus, einfach die Farbe des Körpers, den wir im aktuellen Bildpunkt sehen, auf den Bildschirm zu bringen. Damit könnten wir zwar den Umriß des Körpers erkennen, mehr aber auch nicht. Deshalb gilt es nun festzustellen, wie hell dieser Teil der Körperoberfläche ist.

In unseren ersten Überlegungen sind wir bereits davon ausgegangen, daß eine Oberfläche umso heller erscheint, je senkrechter die Lichtstrahlen darauf treffen. Alles, was wir tun müssen, ist also, den Winkel zu bestimmen, der zwischen dem Normalenvektor auf dem Schnittpunkt und der Geraden vom Schnittpunkt zur Lichtquelle liegt. Ist dieser Winkel Null, so ist die Helligkeit der Oberfläche in diesem Punkt maximal, ist er größer als 90 Grad, so ist dieser Punkt dunkel, da das Licht von "unten" darauf scheint.



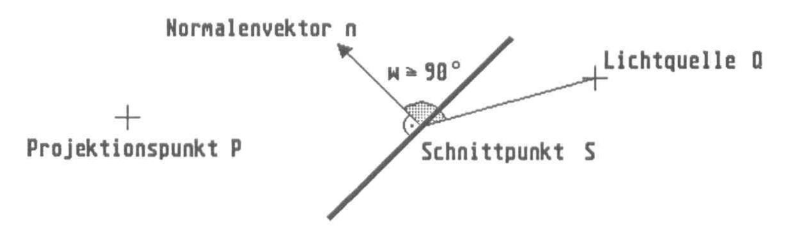


Abbildung 3.11

Auf diese Art und Weise haben wir auch gleich eine einfache Möglichkeit, um festzustellen, ob ein Punkt einer Oberfläche der Lichtquelle zugewandt ist oder nicht. Als Maß für die Helligkeit der Oberfläche nehmen wir den Cosinus des Winkels. Dies hat den Vorteil, daß der Cosinus für senkrechte Lichteinstrahlung (0 Grad) Eins und für extrem flache Lichteinstrahlung (90 Grad) Null ist, so daß wir dies nur mit den drei Farbintensitäten der Oberfläche (rot, grün, blau) multiplizieren müssen, um den richtigen Farbton zu erhalten. Darüber hinaus ist der Cosinus für Winkel größer als 90 Grad negativ, so daß sich unsere Abfrage, ob die Oberfläche der Lichtquelle zugewandt ist, auf einen einfachen Vorzeichentest beschränkt.

In der Natur ist es im übrigen so, daß es keine feste Funktion gibt, die die Helligkeit einer Oberfläche in Abhängigkeit vom Einfallswinkel der Lichtstrahlen berechnet. Vielmehr hat jedes Material seine eigene Funktion. Da wir es hier aber nicht übertreiben wollen, sind wir mit dem Cosinus zufrieden, vor allem, da sich der Cosinus des Winkels zwischen zwei Vektoren sehr einfach berechnen läßt.

Jetzt gilt es nur noch, den Normalenvektor zu ermitteln. Für Ebenen, also auch für Dreiecke, Parallelogramme etc., ist dieser während der gesamten Berechnung fest, so daß wir ihn vor der Berechnung ausrechnen und in Hilf abspeichern können. Der Normalenvektor ergibt sich dabei aus dem Vektorprodukt aus den beiden Vektoren r2 und r3, die ja die Ebene aufspannen. Für die Kugel läßt sich der Normalenvektor noch einfacher berechnen, ist aber leider für jeden Punkt der Kugel verschieden. Dort ist der Normalenvektor nämlich gleich dem Vektor vom Mittelpunkt der Kugel zum Schnittpunkt.

Schwierig wird es natürlich wieder bei Zylinder, Kegel und Ellipsoid. Leider reicht es nicht aus, den Normalenvektor im Raum (r2,r3,r4) auszurechnen und dann zurückzutransformieren. Es gibt jedoch andere Möglichkeiten.

Für den Zylinder liegt der Normalenvektor parallel zur Grundfläche, so daß wir auch nur die Vektoren r2 und r3 zu berücksichtigen brauchen. Die Grundfläche des Zylinders ist (im all-

gemeinen Fall) eine Ellipse, und für die hält die Schulmathematik eine Gleichung bereit:

```
(x^2 / a^2) + (y^2 / b^2) = 1
```

Auflösen nach y liefert:

```
y = b * SQR(1- (x^2 / a^2))
y = b/a * SQR(a^2 - x^2)
```

Diese Gleichung leiten wir nun nach y ab und erhalten damit die Steigung der Ellipse:

```
y' = -b/a * x/SQR(a^2 - x^2)
```

Wenn wir y' als Steigung einer Geraden betrachten, so ist die Senkrechte dazu unser gewünschter Normalenvektor. Die Senkrechte erhalten wir, indem wir den negativen Kehrwert der Steigung bilden. Die Steigung der Senkrechten entspricht dabei ny/nx, also der Steigung des Normalenvektors:

```
ny/nx = a/b * SQR(a^2 - x^2)/x
```

Dieses spalten wir in je eine Gleichung für nx und ny auf, so daß sich ny/nx nicht verändert:

```
ny = (a*b)*SQR(a^2 - x^2)
nx = (b^2)*x
```

Das sieht im Moment noch ziemlich willkürlich aus, löst sich gleich aber vorzüglich auf. Nun setzen wir die ursprüngliche Ellipsengleichung in die Formel für ny ein, um die Wurzel aufzulösen und erhalten für ny:

```
ny = (a*b)*(y*a/b)  = >
ny = (a^2)*y
```

Jetzt haben wir zwei wunderschöne Gleichungen für nx und ny, nur mit dem Nachteil, daß wir in unserer Zylindergleichung weder a, b, x noch y haben. Deshalb müssen wir nun festlegen, welcher Parameter welchem entspricht. a ist die Hauptachse der

Ellipse, entspricht daher der Länge des Vektors r2, also |r2|. b ist die andere Achse und entspricht der Länge von r3, also |r3|. x und y sind die Koordinaten eines Punktes, welche bei uns u*r2 und v*r3 heißen, wobei u und v die Schnittpunktsparameter sind. Da unsere Parameter Vektoren sind und verschiedene Richtungen haben, können wir die Gleichungen für nx und ny gefahrlos addieren. Das Ergebnis ist der Normalenvektor auf den Zylinder:

```
n = |r3|^2 * u*r2 + |r2|^2 * v*r3
```

Da sich hier die Quadrate der Längen von r2 und r3 während der Berechnung nicht ändern, legen wir auch diese wieder in unser Feld Hilf, das uns hilft, enorm viel Zeit zu sparen.

Beim Kegel sieht die Sache ganz ähnlich aus, nur daß der Normalenvektor in Richtung der Kegelspitze gebogen ist. Wir bilden zuerst wieder den Normalenvektor auf die Grundflächenellipse. Dann ändern wir die Länge des Normalenvektors so, daß sie genauso groß ist wie die Länge von u*r2+v*r3. Nun verhält sich die Länge von r4 zur Länge des Normalenvektors exakt so wie die Steigung der Kegeloberfläche, wenn wir einen Längsschnitt des Kegels betrachten. Den Normalenvektor auf diesen Punkt der Kegeloberfläche erhalten wir wieder durch Berechnung der Senkrechten. Sei n2 der Normalenvektor auf die Grundfläche, wie er oben für den Zylinder bereits beschrieben wurde. So ist

```
n1 = n2/|n2| * |u*r2 + v*r3|
```

der Normalenvektor auf die Grundfläche, aber mit der Länge von u*r2+v*r3. Die Richtung hat sich dabei nicht verändert! Der Normalenvektor auf die Kegeloberfläche ergibt sich dann mit:

```
n = |n1|/|r4| * r4 + |r4|/|n1| * n1
```

Die Herleitung des Normalenvektors für das Ellipsoid möchte ich den Spezialisten unter Ihnen überlassen, dies würde an dieser Stelle etwas langwierig (und -weilig) wirken. Daher sei hier nur kurz die Lösung erwähnt:

```
n = (|r3|^2 + |r4|^2) * u*r2
  + (|r2|^2 + |r4|^2) * v*r3
  + (|r2|^2 + |r3|^2) * w*r4
```

Nun können wir also die Helligkeit jedes Punktes der Oberfläche eines unserer Grundobjekte bestimmen, wenn wir zuvor festgelegt haben, wo die Lichtquelle liegt. Ganz besonders schön wäre es natürlich, wenn wir auch den Lichtreflex der Lichtquelle in unserer Oberfläche sehen könnten. Dies läßt sich nun glücklicherweise für alle Grundobjekte allgemeingültig lösen. Dazu berechnen wir den gespiegelten Lichtstrahl und dann den Winkel zwischen diesem und dem Sehstrahl, also den Vektor vom Schnittpunkt zum Projektionspunkt. Wenn dieser Winkel gleich Null ist, so sehen wir direkt in die Lichtquelle, und dementsprechend ist der Lichtreflex sehr stark. Je größer der Winkel, desto geringer ist dann der Lichtreflex.

Um den gespiegelten Lichtstrahl zu erhalten, berechnen wir zuerst den Abstand der Lichtquelle Q von der Ebene, die tangential zur Oberfläche durch den Schnittpunkt S verläuft. Nun multiplizieren wir den Normalenvektor n mit einem Faktor, so daß dessen Länge gleich dem Abstand von Q zur Tangentialebene ist. Danach läßt sich der gespiegelte Vektor SP wie folgt bestimmen:

```
SP = (S-Q) + 2*n
```

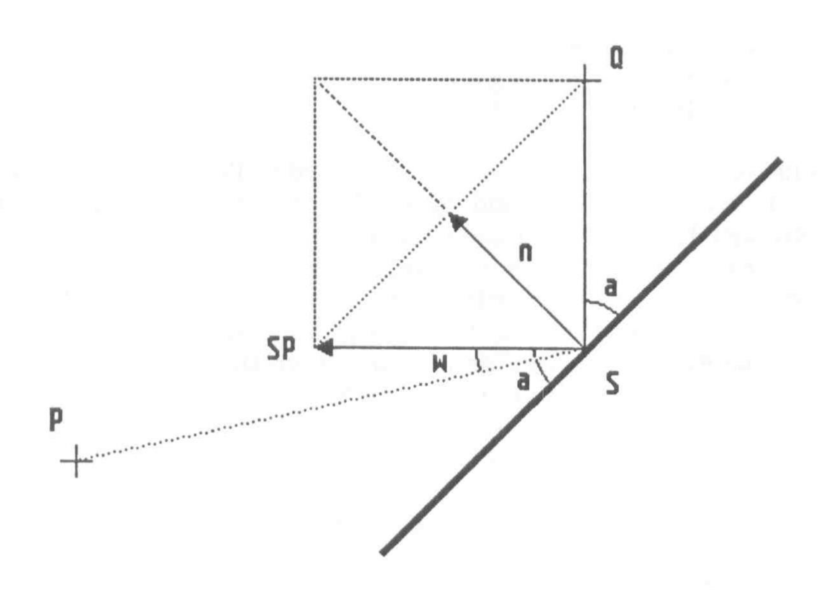


Abbildung 3.12

Statt des Winkels verwenden wir wieder den Cosinus davon und speichern ihn in der Variablen Spiegel ab. Damit der Lichtreflex nicht zu großflächig wird, potenzieren wir das Ergebnis noch mit einer relativ großen Zahl, so daß der Wert von Spiegel bereits für kleine Winkel stark gegen Null geht.

Nachdem wir nun endlich alle Berechnungen zusammen haben, um die Helligkeit eines Punktes einer Oberfläche bestimmen zu können, packen wir sie in ein Unterprogramm und geben ihm den bezeichnenden Namen BestimmeHell. Dort wird sowohl die Helligkeit der Oberfläche in der Variablen Hell abgelegt als auch die Spiegelung der Lichtquelle in Spiegel. Dieses Unterprogramm hat mit der Farbe des Körpers noch nichts zu tun, diese kommt erst später an die Reihe.

```
SUB Bestimmehell(n%,Px,Py,Pz,Spiegel,Original!) STATIC
SHARED Hilf(),K(),Mat(),Qx,Qy,Qz,Ac,Bc,Cc,Sx,Sy,Sz
SHARED Nx,Ny,Nz,Nl,Spx,Spy,Spz,Hell
  '  = > hell = Helligkeit, SP = gespiegelter Vektor,
  ' Spiegel = Intensität der LQ-Spiegelung

  IF K(n%,0,0)< = 9 THEN
      ' Bestimme gespiegelten Vektor SP
    Nx = Hilf(n%,0)
    Ny = Hilf(n%,1)
    Nz = Hilf(n%,2)
    IF Original!<>True THEN
      IF FN CosinWinkel(Nx,Ny,Nz,Px-Sx,Py-Sy,Pz-Sz)<0 THEN
        'Normalenvektor muß zum Projektionspunkt zeigen!
        Nx = -Nx
        Ny = -Ny
        Nz = -Nz
      END IF
    END IF
    Dqe = Nx*(Qx-Sx)+Ny*(Qy-Sy)+Nz*(Qz-Sz)

    Spx = Sx-Qx+2*Dqe*Nx
    Spy = Sy-Qy+2*Dqe*Ny
    Spz = Sz-Qz+2*Dqe*Nz

    Spiegel = FN CosinWinkel(Spx,Spy,Spz,Px-Sx,Py-Sy,Pz-Sz)

    IF Spiegel<0 THEN
      Spiegel = 0
    END IF

    Hell = FN CosinWinkel(Nx,Ny,Nz,Qx-Sx,Qy-Sy,Qz-Sz)
  ELSE
      ' Bestimme gespiegelten Vektor SP
    IF K(n%,0,0) = 10 THEN
      Nx = Sx-K(n%,1,0)
      Ny = Sy-K(n%,1,1)
      Nz = Sz-K(n%,1,2)
    END IF

    IF K(n%,0,0)> = 20 AND K(n%,0,0)<24 THEN
      Nx = Hilf(n%,4)*Ac+Hilf(n%,1)*Bc
      Ny = Hilf(n%,5)*Ac+Hilf(n%,2)*Bc
      Nz = Hilf(n%,6)*Ac+Hilf(n%,3)*Bc
```

```
  IF K(n%,0,0)> = 22 THEN
    Nl = SQR(Nx*Nx+Ny*Ny+Nz*Nz)
    IF Nl<>0 THEN
      a = (Ac*K(n%,2,0)+Bc*K(n%,3,0))^2
      a = a+(Ac*K(n%,2,1)+Bc*K(n%,3,1))^2
      a = (SQR(a+(Ac*K(n%,2,2)+Bc*K(n%,3,2))^2))/Nl
      Nx = Nx*a
      Ny = Ny*a
      Nz = Nz*a
      b = SQR(Nx*Nx+Ny*Ny+Nz*Nz)/Hilf(n%,7)
      Nx = K(n%,4,0)*b+Nx/b
      Ny = K(n%,4,1)*b+Ny/b
      Nz = K(n%,4,2)*b+Nz/b
    ELSE
      ' falls Spitze des Kegels:
      Nx = K(n%,4,0)
      Ny = K(n%,4,1)
      Nz = K(n%,4,2)
    END IF
  END IF
END IF

IF K(n%,0,0)> = 24 THEN
  Nx = Ac*Hilf(n%,1)+Bc*Hilf(n%,4)+Cc*Hilf(n%,7)
  Ny = Ac*Hilf(n%,2)+Bc*Hilf(n%,5)+Cc*Hilf(n%,8)
  Nz = Ac*Hilf(n%,3)+Bc*Hilf(n%,6)+Cc*Hilf(n%,9)
END IF

IF FN CosinWinkel(Nx,Ny,Nz,Px-Sx,Py-Sy,Pz-Sz)<0 THEN
    ' = > Normalenvektor zeigt zum Projektionspunkt
  Nx = -Nx
  Ny = -Ny
  Nz = -Nz
END IF
Nl = SQR(Nx*Nx+Ny*Ny+Nz*Nz)
Nx = Nx/Nl
Ny = Ny/Nl
Nz = Nz/Nl

Dqe = Nx*(Qx-Sx)+Ny*(Qy-Sy)+Nz*(Qz-Sz)

Spx = Sx-Qx+2*Dqe*Nx
Spy = Sy-Qy+2*Dqe*Ny
Spz = Sz-Qz+2*Dqe*Nz

Spiegel = FN CosinWinkel(Spx,Spy,Spz,Px-Sx,Py-Sy,Pz-Sz)
```

```
    IF Spiegel<0 THEN
      Spiegel = 0
    END IF

    Hell = FN CosinWinkel(Nx,Ny,Nz,Qx-Sx,Qy-Sy,Qz-Sz)
  END IF

  IF Mat(K(n%,0,2),4)>0 THEN
    Spiegel = 1.5*Spiegel^(30*Mat(K(n%,0,2),4))
  END IF
END SUB
```

Die Variablen Ac, Bc und Cc enthalten die Schnittpunktsparameter für u, v und w, falls es sich bei dem Objekt um einen Zylinder, Kegel oder Ellipsoid handelt. Der Normalenvektor wird normiert, hat also beim Verlassen des Unterprogramms die Länge Eins. Dies ist wichtig für spätere Berechnungen. Dqe entspricht dem Abstand der Lichtquelle zur Tangentialebene. Die Formel für die Berechnung von Spiegel, welche direkt am Ende des Unterprogramms steht, ist reine Willkür und wurde durch Ausprobieren ermittelt. Wenn man eine matte Oberfläche definiert hat, also Mat(n,4) relativ klein ist, so ist auch der Lichtreflex der Lichtquelle entsprechend gering.

3.4.3 Die Schattenseiten der Welt

Nein, keine Panik! Es kommt jetzt kein Bericht über die Abgründe menschlicher Unmenschlichkeit, sondern es geht schlicht und einfach darum, unseren berechneten Bildern durch Schatten zu mehr Realität und Plastizität zu verhelfen.

Da wir bereits umgangssprachlich formuliert haben, wie wir dabei vorgehen wollen, können wir auch gleich zur Sache kommen. Gegeben haben wir den Schnittpunkt S des Sehstrahls mit einem Objekt und die Position der Lichtquelle Q. Alles, was wir tun müssen, ist zu prüfen, ob sich die Gerade von S nach Q mit irgendeinem Objekt schneidet und ob dieser Schnittpunkt zwischen S und Q liegt und nicht davor oder dahinter.

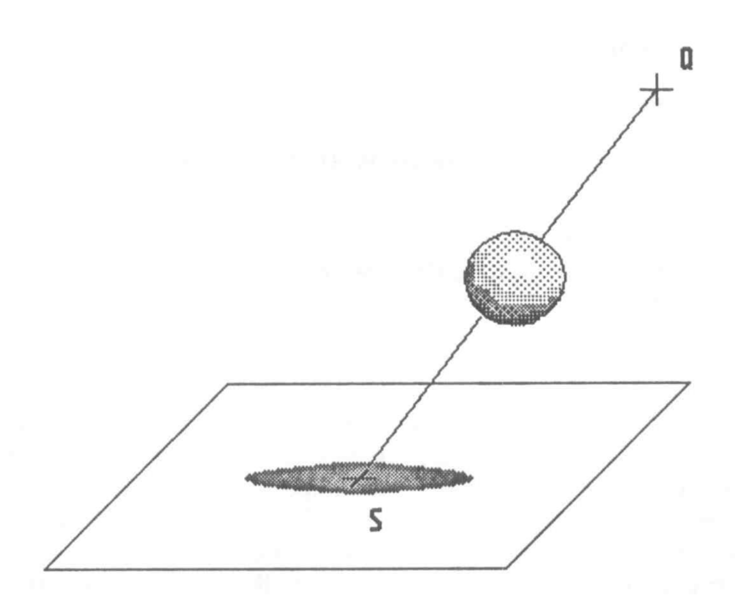


Abbildung 3.13

Als Richtungsvektor der Geraden verwenden wir Q-S. Wegen der Ungenauigkeit beim Umgang mit reellen Zahlen fragen wir bei der Schnittpunktberechnung ab, ob der Parameter l der Geraden größer als eine Schwelle ist, die von der Ungenauigkeit der Zahlendarstellung abhängt; wir sprachen bereits darüber. Wenn wir dies nicht beachten würden, so würde im ungünstigsten Fall wieder das selbe Objekt, auf dem der Schnittpunkt S liegt, als "Schattenspender" berechnet werden, nur weil bei den Berechnungen durch die Ungenauigkeit nicht 0 sondern 0.00005 herausgekommen ist.

Da es in unserer Computerwelt kein Streulicht gibt, wäre der Schatten natürlich tiefschwarz. Weil dies aber nicht immer gewünscht ist, können wir für jedes Material eine Schattenhelligkeit angeben, die zwischen Null und Eins liegt und die angibt, wie hell die Oberfläche des Materials im Schatten ist.

In unserem Programm ist die Abfrage extrem einfach, da wir schon ein Unterprogramm haben, das den Schnittpunkt zwischen einer Geraden und allen Objekten berechnet. Wir brauchen dieses nur nochmals aufzurufen, nur mit dem Unterschied, daß die Gerade diesmal vom Schnittpunkt zur Lichtquelle läuft. Wenn wir als Ergebnis einen Schnittpunkt erhalten, so liegt der entsprechende Punkt im Schatten, andernfalls können wir ihn ganz normal einfärben. Das Programmstück könnte wie folgt aussehen:

```
  'Befindet sich am aktuellen Bildpunkt überhaupt was?
  CALL WelcherKoerper(0,Px,Py,Pz,Rx,Ry,Rz,True,False)

IF Koerper% > 0 THEN

  'Helligkeit des Punktes bestimmen:
  CALL Bestimmehell(Koerper%,Px,Py,Pz,Spiegel,Original!)
  RestHell(StackPtr%) = Mat(K(Koerper%,0,2),4)
  'Spieglungsfaktor

  IF Hell< = 0 THEN
      ' nicht beleuchtet:
    Hell.r = Mat(K(Koerper%,0,2),0)*Mat(K(Koerper%,0,2),3)
    Hell.g = Mat(K(Koerper%,0,2),1)*Mat(K(Koerper%,0,2),3)
    Hell.b = Mat(K(Koerper%,0,2),2)*Mat(K(Koerper%,0,2),3)
  ELSE

  'Liegt der Punkt vielleicht im Schatten?

  'Die folgende Anweisung ist notwendig, damit durch die
  'Rechenungenauigkeit nicht eventuell wieder der gleiche
  'Körper herauskommt!

    Kp% = Koerper%
    CALL WelcherKoerper(Kp%,Sx,Sy,Sz,Qx-Sx,Qy-Sy,Qz-Sz,False,True)
    SWAP Koerper%,Kp%

    IF Kp%>0 AND la<1 THEN
      ' im Schatten:
      Hell.r = Mat(K(Koerper%,0,2),0)*Mat(K(Koerper%,0,2),3)
      Hell.g = Mat(K(Koerper%,0,2),1)*Mat(K(Koerper%,0,2),3)
      Hell.b = Mat(K(Koerper%,0,2),2)*Mat(K(Koerper%,0,2),3)
    ELSE
```

```
        Hell = Hell*(1-RestHell(StackPtr%))
        '
        IF Hell<Mat(K(Koerper%,0,2),3) THEN
    'Dunkler als der Schatten darf kein Punkt der Oberfläche sein,
    'sonst wirkt es unrealistisch!
          Hell = Mat(K(Koerper%,0,2),3)
        '
        END IF
        ' Spiegelung der Lichtquelle auf Oberfläche:
        Hell.r =
Hell*Mat(K(Koerper%,0,2),0)+Spiegel*RestHell(StackPtr%)
        Hell.g =
Hell*Mat(K(Koerper%,0,2),1)+Spiegel*RestHell(StackPtr%)
        Hell.b =
Hell*Mat(K(Koerper%,0,2),2)+Spiegel*RestHell(StackPtr%)

      END IF
    END IF
  END IF
```

Nun müßten wir nur noch den Punkt des Bildschirms entsprechend der Werte von Hell.r (rot), Hell.g (grün) und Hell.b (blau) einfärben. Aber wir wollen noch einen Schritt weiter gehen. Ich hoffe, Sie sind von den vielen Schritten noch nicht allzu müde, denn jetzt wird es erst richtig interessant.

3.4.4 Spieglein, Spieglein...

...an der Wand, wer berechnet die schönsten Bilder im ganzen Land? "Frau Königin, ich sag's Euch ins Gesicht: Ihr beherrscht die Spiegelung nicht!" Daraufhin schickte die Königin ihren Jäger in die nächste Buchhandlung, daß er das Buch "3-D-Grafik-Programmierung" kaufen solle.

Und die Moral von der Geschicht': Ohne Spiegel ist's nur halb so schön.

Deshalb wollen wir als letztes Feature die Spiegelung in Angriff nehmen. Diesmal geht es nicht ganz so einfach wie bisher. Zwar können wir den gespiegelten Sehstrahl berechnen und auch des-

sen Schnittpunkt mit einem Objekt, es stellt sich aber das Problem, in welcher Art und Weise wir die verschiedenen Farben der verschiedenen Oberflächen miteinander verknüpfen.

Je matter eine Oberfläche ist, desto stärker spielt die Farbe dieser Oberfläche eine Rolle und die Farben der sich in dieser Oberfläche spiegelnden Objekte sind kaum zu sehen. Ferner sind die Farben der Oberfläche am schwächsten die sich zuletzt spiegelt, die der Sehstrahl also als letzte erreicht. In der Natur ist es ja schließlich auch so, daß der Lichtstrahl bei jeder Spiegelung an Intensität verliert, so daß seine ursprüngliche Farbinformation immer mehr verändert wird.

Daher werden wir wie folgt vorgehen: Zuerst berechnen wir sämtliche Schnittpunkte und deren Farbe, wie dies im obigen Programmstück bereits beschrieben wurde. Diese Farben legen wir auf dem Stack ab, welchen wir uns aus einem Feld zusammenbsteln. Nach der letzten Spiegelung nehmen wir die berechnete Farbe und verknüpfen sie mit der letzten Farbe. Das Resultat verknüpfen wir mit der vorletzten Farbe usw., bis wir bei der ersten Farbe angekommen sind.

```
Berechnepunkt:
  ' = >Berechne Helligkeit des Punktes  = > stack
  StackPtr% = StackPtr% + 1
  CALL WelcherKoerper(0,Px,Py,Pz,Rx,Ry,Rz,Original!,False)
  IF Koerper% > 0 THEN
    CALL Bestimmehell(Koerper%,Px,Py,Pz,Spiegel,Original!)
    RestHell(StackPtr%) = Mat(K(Koerper%,0,2),4)

    IF Hell< = 0 THEN
        ' nicht beleuchtet:
      Hell.r = Mat(K(Koerper%,0,2),0)*Mat(K(Koerper%,0,2),3)
      Hell.g = Mat(K(Koerper%,0,2),1)*Mat(K(Koerper%,0,2),3)
      Hell.b = Mat(K(Koerper%,0,2),2)*Mat(K(Koerper%,0,2),3)
    ELSE
      Kp% = Koerper%
        'Schatten ?
      CALL WelcherKoerper(Kp%,Sx,Sy,Sz,Qx-Sx,Qy-Sy,Qz-Sz,False,True)
      SWAP Koerper%,Kp%
      IF Kp%>0 AND la<1 THEN
          ' im Schatten:
```

```
      Hell.r = Mat(K(Koerper%,0,2),0)*Mat(K(Koerper%,0,2),3)
      Hell.g = Mat(K(Koerper%,0,2),1)*Mat(K(Koerper%,0,2),3)
      Hell.b = Mat(K(Koerper%,0,2),2)*Mat(K(Koerper%,0,2),3)
    ELSE
        ' Spiegelung der Lichtquelle auf Oberfläche:
      Hell = Hell*(1-RestHell(StackPtr%))
      IF Hell<Mat(K(Koerper%,0,2),3) THEN
        Hell = Mat(K(Koerper%,0,2),3)
      END IF
      Hell.r = Hell*Mat(K(Koerper%,0,2),0)+Spiegel*RestHell(StackPtr%)
      Hell.g=Hell*Mat(K(Koerper%,0,2),1)+Spiegel*RestHell(StackPtr%)
      Hell.b=Hell*Mat(K(Koerper%,0,2),2)+Spiegel*RestHell(StackPtr%)
    END IF
  END IF

  IF (RestHell(StackPtr%)>0) AND (StackPtr%<MaxStack%) THEN
      ' Bestimme Spiegelung auf Oberfläche
      ' Bestimme gespiegelten Vektor zu P-S:
    Dqe=Nx*(Px-Sx)+Ny*(Py-Sy)+Nz*(Pz-Sz)
    Rx=Sx-Px+2*Dqe*Nx
    Ry=Sy-Py+2*Dqe*Ny
    Rz=Sz-Pz+2*Dqe*Nz

    Stack(StackPtr%,0)=Hell.r
    Stack(StackPtr%,1)=Hell.g
    Stack(StackPtr%,2)=Hell.b

    Px = Sx
    Py = Sy
    Pz = Sz
    Original! = False
    GOSUB Berechnepunkt      ' Rekursion !!!!

    Hell.r=Stack(StackPtr%,0)+Stack(StackPtr%+1,0)*RestHell(StackPtr%)
    Hell.g=Stack(StackPtr%,1)+Stack(StackPtr%+1,1)*RestHell(StackPtr%)
    Hell.b=Stack(StackPtr%,2)+Stack(StackPtr%+1,2)*RestHell(StackPtr%)
  END IF
ELSE
  IF StackPtr% = 0 THEN
    Hell.r=-1
    Hell.g=-1
    Hell.b=-1
  ELSE
    Hell.r=0
    Hell.g=0
    Hell.b=0
```

```
    END IF
  END IF
  Stack(StackPtr%,0)=Hell.r
  Stack(StackPtr%,1)=Hell.g
  Stack(StackPtr%,2)=Hell.b
  RestHell(StackPtr%)=0
  StackPtr% = StackPtr%-1
RETURN
```

Dieses Routine enthält auch den Programmteil, der schon beim Schatten besprochen wurde. Die verwendeten Variablen wurden dort bereits besprochen, so daß hier nur zu erwähnen wäre, daß StackPtr% die Berechnungstiefe angibt. Im Hauptprogramm, aus dem Berechnepunkt aufgerufen wird, wird StackPtr% auf -1 gesetzt. Bei jeder Spiegelung wird dieser dann um Eins erhöht, bis keine Spiegelung mehr vorliegt oder die maximale Stacktiefe (20) erreicht ist.

3.4.5 Maler Klecksel oder: Wie bringe ich die Farbe auf den Bildschirm

Das Ergebnis unserer bisherigen Berechnungen sind drei Werte, die den Rot-, Grün- und Blauton der Farbe bestimmen und die wir nun irgendwie auf den Bildschirm bringen müssen. Da wir nicht beliebig viele Farben zur Verfügung haben, sondern maximal 64 verschiedene, müssen wir uns jetzt überlegen, wie wir diese so zusammenmischen, daß wir möglichst die gesuchte Farbe erhalten.

"Halt!" höre ich Sie rufen, "im HAM-Modus haben wir doch alle 4096 Farben zur Verfügung, und das sind doch mehr als genug." Das stimmt schon, nur kann ein beliebiger Pixel trotzdem nur eine von 64 Farben annehmen: 16 Grundfarben plus 16 Farben durch Ändern der Rotintensität plus 16 Farben durch Ändern der Grünintensität plus 16 Farben durch Ändern der Blauintensität. Macht zusammen: 64 Farben, dabei sind allerdings noch drei doppelt.

Also am Mischen kommen wir nicht vorbei. Wir mischen, indem wir Muster definieren und diese bei der Ausgabe benutzen. Das Muster selbst ist zweifarbig: Eine Farbe ist durch die Vordergrundfarbe und die andere durch die Hintergrundfarbe festgelegt. Um nun verschiedene Mischungen zu bekommen, definieren wir verschiedene Muster, mal mit mehr, mal mit weniger Punkten von einer Farbe. Das Muster wird von der Betriebssystemroutine zum Ausgeben eines Rechtecks automatisch benutzt, wenn man in der RastPort-Struktur einen Zeiger darauf setzt.

Allerdings stellt sich uns nun wieder ein Problem in den Weg, welches wir nicht in guter alter Wildwestmanier über den Haufen schießen können: Welche zwei Farben nehmen wir, die gemischt unsere Wunschfarbe ergeben sollen? So einfach, wie sich das zunächst angehört hat, ist es nämlich nicht. Wenn man das Bild nur in Grautönen berechnen lassen würde, reichte es aus, die beiden Grautöne zu nehmen, die am nächsten an dem gesuchten liegen, nicht aber bei Farbe.

Stellen Sie sich Farbe dreidimensional vor: Eine Rot-Achse, eine Grün-Achse und eine Blau-Achse. Die Menge aller denkbaren Farben wäre dann ein Würfel, der mit einer Ecke am Ursprung hängen würde. Um Ihnen die Problematik anschaulicher zu machen, hier ein Beispiel:

Sie wollen einen Farbverlauf der Farbe Gelb darstellen. Da Sie mit Speicherplatz geizen müssen, hat Ihr Bildschirm nur vier Farben: Schwarz, Rot, Grün und Gelb. Rot und Grün brauchen Sie, da das zu berechnende Bild auch rote und grüne Objekte enthält. Wenn man die Farben in ein Koordinatensystem einzeichnet, welches für diesen einfachen Fall ruhig zweidimensional sein kann, ergibt sich folgendes Bild:

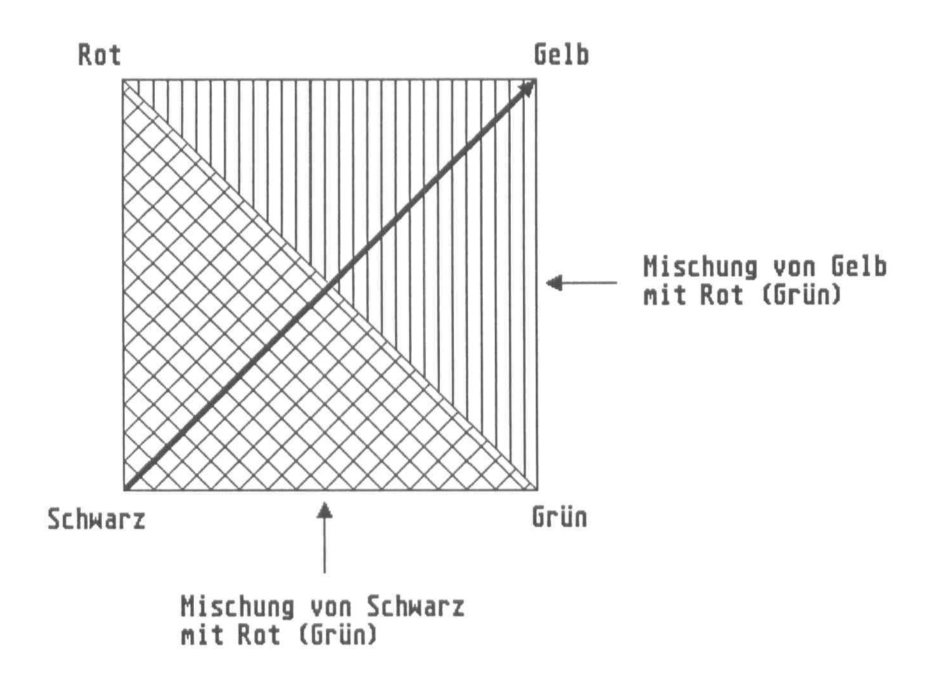


Abbildung 3.14

Wenn die Gelbintensität kleiner als 0.5 ist, so wird als am besten passende Farbe Schwarz gefunden. Am zweitbesten paßt dann aber Rot (oder Grün), da dies näher liegt als Gelb, so daß sich ein Verlauf von Schwarz nach Rot (Grün) ergibt. Steigt die Gelbintensität dann über 0.5, so paßt endlich Gelb am besten. Am zweitbesten paßt aber nicht Schwarz, sondern wieder Rot (oder Grün), so daß sich diesmal ein Verlauf von Rot (Grün) nach Gelb ergibt. Ob Rot oder Grün gewählt wird, ist in diesem Fall ziemlich egal, da beide Farben den gleichen Abstand von der gewünschten Farbe haben. Ändern läßt sich das Ergebnis nur durch eine Erhöhung der Anzahl der Farben.

Eine unattraktive Lösung wäre es, wenn der Algorithmus sämtliche Möglichkeiten durchprobieren und die am besten passende nehmen würde. Bei 64 Farben und 25 Mustern bräuchte die Routine selbst in Assembler fünf Sekunden pro Pixel, für ein

320x200-Bild insgesamt mehr als drei Tage! Wie lange ein entsprechendes BASIC-Programm brauchen würde, alleine um die Pixel zu setzen, wage ich gar nicht auszurechnen.

Nächste Idee: Sie suchen die beiden Farben, deren Farbton dem gesuchten am ähnlichsten ist. Dazu müssen Sie die Richtung des Farbvektors bestimmen, also seine Polarkoordinaten. Nachteil: Auch das Umrechnen der gesuchten Farbe in Polarkoordinaten kostet Zeit. Außerdem entspricht das Ergebnis immer noch nicht ganz dem gewünschten. So wird, um einen Grauton zu bekommen, beispielsweise Weiß und Schwarz gemischt statt zweier Grautöne. Hier hängt sehr viel von den Mustern selbst ab, nämlich in welchem Verhältnis die beiden Farben gemischt werden.

Ich muß zu meiner Schande gestehen, daß ich den optimalen Algorithmus noch nicht gefunden habe. Verwendet habe ich den ersten, der die beiden am nächsten an der gesuchten Farbe liegenden Farben nimmt und diese zusammenmischt. Wenn man die Grundfarben gut verteilt, so erzielt man damit, vor allem im HAM-Modus, trotzdem gute Ergebnisse. Vielleicht fällt mir ja mal was besseres ein, oder wie wär's mit Ihnen? Denken Sie sich doch mal einen guten und schnellen Algorithmus aus.

Da BASIC nicht gerade die schnellste Sprache ist, habe ich den Algorithmus in Assembler realisiert. Besser wäre es natürlich gewesen, wenn das ganze Programm in Assember geschrieben wäre, allerdings ist die Lesbarkeit von Assemblerprogrammen nicht besonders hoch. Außerdem ist die Länge des Programms ziemlich abschreckend.

Geschrieben ist die Assemblerroutine mit PROFIMAT. Der Quelltext befindet sich ebenfalls auf der Diskette (SetPoint.ASM). Wenn Sie diesen selber übersetzen wollen, so müssen Sie bei PROFIMAT PC-relativen Code erzeugen lassen. Ferner müssen Sie sich die Adressen der drei Unterprogramme und der Variablen RastPort, Modus, MaxFarben, Farben, RasterW und RasterH rausschreiben, und diese in das BASIC-Unterprogramm InitSetPoint einbauen. Abspeichern müssen Sie

den erzeugten Code als "SetPoint.B". Geladen wird die Routine mit InitSetPoint. Verändert werden die Variablen bei RasterInit und Farbpalette. Aufgerufen wird die Routine mit:

```
CALL OSSetPoint&(x1,y1,x2,y2,rot*1024,gruen*1024,blau*1024)
```

wobei (x1,y1) die obere linke Ecke und (x2,y2) die untere rechte Ecke des zu zeichnenden Rechtecks sind. Die Werte für rot, gruen und blau müssen zwischen 0 und 1024 liegen. Übergeben werden im übrigen nur ganzzahlige Werte.

Voraussetzung für das Funktionieren von OSSetPoint ist, das die Rot-, Grün- und Blauwerte der Grundfarben im Feld OSFarben abgelegt sind. Dieses Feld befindet sich direkt hinter der Assemblerroutine, und die Werte müssen hineingepoked werden. Jedes Element enthält dabei vier Werte (16 Bit): Nummer des zugehörigen Farbregisters, Rotintensität*1024, Grünintensität*1024 und Blauintensität*1024. Die Anzahl der Grundfarben muß in OSMaxFarben stehen. OSRastPort enthält einen Zeiger auf den RastPort des Fensters oder Bildschirms, OSModus den Darstellungsmodus, also z. B. $800 für HAM-Modus. OSRasterW und OSRasterH enthalten Breite und Höhe des Fensters/Bildschirms, da OSSetPoint ein eigenes Clipping vornimmt.

Um gute Farbverläufe hinzubekommen, sollten Sie die Farbpalette für jedes Bild optimal einstellen. Lassen Sie das Bild dazu zunächst in grober Auflösung durchrechnen, und merken Sie sich die Farben, die darin vorkommen. Dann stellen Sie die Farbpalette so ein, daß für jede Farbe möglichst viele Farbstufen vorkommen. Am besten ist es, wenn Sie den Hold-And-Modify-Modus benutzen, allerdings sollten Sie sich auch da die Mühe machen, und die Farbpalette optimal einstellen.

3.5 Optimierungen

So, wie wir den Algorithmus bis jetzt formuliert haben, funktioniert er zwar, aber seine Laufzeit... Daher sind wir gut beraten, wenn wir uns Gedanken darüber machen, was sich an dem Algorithmus noch verbessern läßt.

Eine Sache, nämlich das Ausrechnen und Merken von Ausdrücken, die sich während der Berechnung nicht ändern, haben wir ja bereits eingebaut. Das Unterprogramm, das diese Vor-Berechnungen für uns ausführt, heißt InitSchattiere. Die berechneten Werte werden im Feld Hilf abgelegt, dessen Elemente für verschiedene Objekte auch verschiedene Bedeutung haben. So enthält Hilf für Ebenen unter anderem den normierten Normalenvektor, für Zylinder, Kegel und Ellipsoid die Nennerdeterminante für die Basistransformation. Eine vollzählige Aufzählung sparen wir uns hier, da dies etwas zu weit führen würde.

Die nächste Stufe der Optimierung betrifft die Schnittpunktsberechnung. Betrachten Sie dazu folgende Abbildung, welche den Bildschirm darstellen soll, mit der fertigen Szene:

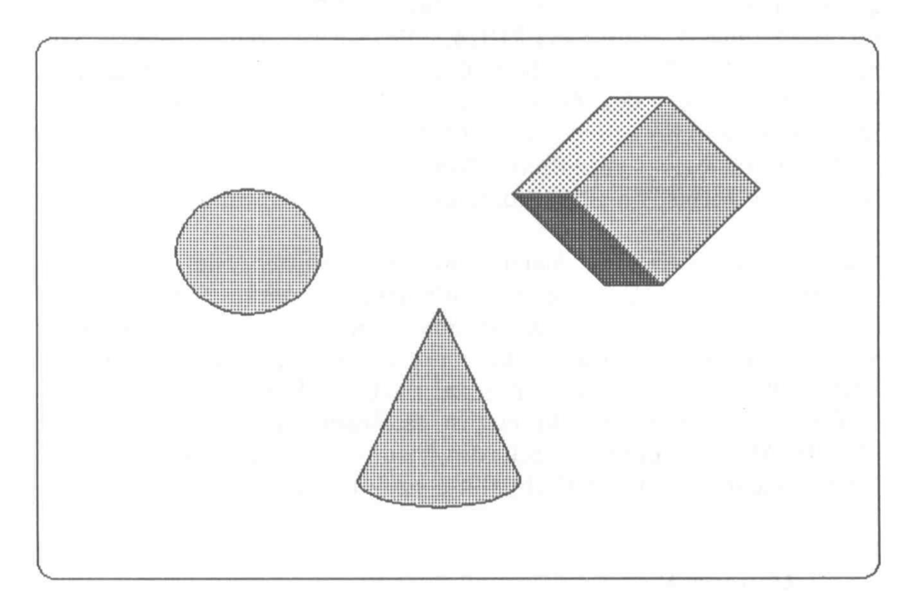


Abbildung 3.15

Die Objekte nehmen einen relativ geringen Teil des Bildschirms ein. Wir könnten als erstes ein minimales Rechteck festlegen, in dem sämtliche Objekte liegen. Dann bräuchte unser Programm nur noch für alle die Punkte, die innerhalb dieses Rechtecks liegen, einen Sehstrahl loszuschicken.

Als nächstes legen wir für jedes einzelne Objekt ein minimales Rechteck fest, so daß das Objekt komplett darin liegt, was die Darstellung auf dem Bildschirm anbetrifft. In unserem Unterprogramm WelcherKoerper fragen wir dann für jedes Objekt ab, ob der aktuelle Sehstrahl durch dieses minimale Rechteck läuft, und nur wenn ja, berechnen wir den Schnittpunkt. Diese Abfrage, ob der aktuelle Bildschirmpunkt innerhalb eines Rechtecks liegt, geht wesentlich schneller vonstatten, als die rechenaufwendige Schnittpunktsberechnung, die schlimmstenfalls mehrere Determinantenberechnungen ausführen muß (Zylinder, Kegel, Ellipsoid).

Die Rechtecke für unser obiges Bild sehen dann zum Beispiel wie folgt aus:

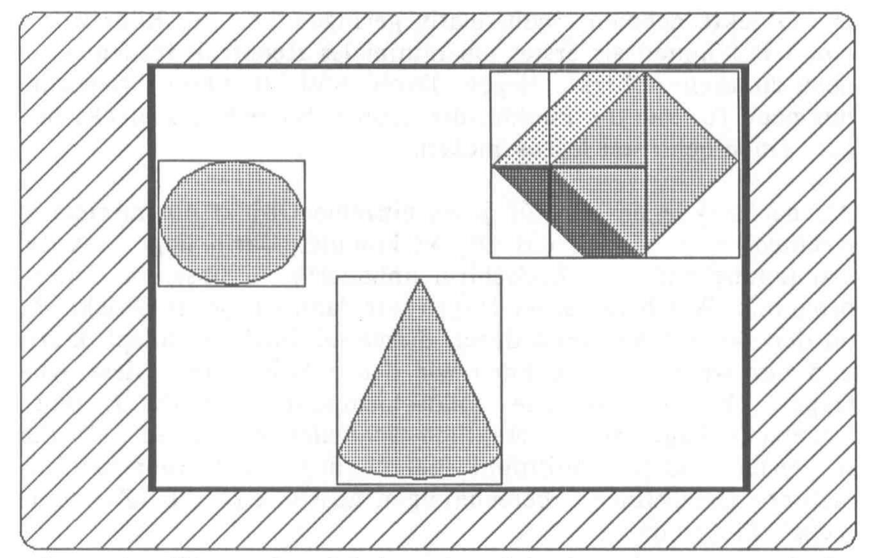


Abbildung 3.16

Die Rechtecke für die einzelnen Objekte sind dabei dünn, das Rechteck für den Bildausschnitt dick gezeichnet. Beachten Sie dabei bitte, daß der Würfel aus sechs Parallelogrammen zusammengesetzt ist, wobei hier aber nur drei gezeichnet wurden, da sonst die Abbildung zu unübersichtlich geworden wäre.

Ablegen können wir die Rechtecksgrenzen wieder in einem Feld, sagen wir in MinMax. Jedes Element von MinMax enthält dabei folgende Werte:

```
MinMax(n,0) = minimale X-Koordinate für Objekt n.
MinMax(n,1) = minimale Y-Koordinate für Objekt n.
MinMax(n,2) = maximale X-Koordinate für Objekt n.
MinMax(n,3) = maximale Y-Koordinate für Objekt n.
```

Das Unterprogramm, welches diese Rechtecke errechnet, hört auf den Namen InitMinMax. Für "runde" Objekte, also Kreisfläche, Zylinder, Kugel etc. wird ein Rechteck bzw. ein Quader

um das Objekt herumgelegt, und dessen Eckpunkte bilden dann die Basis für die Rechtecksberechnung. Hier können Sie also noch fleißig optimieren, indem Sie zum Beispiel ein Achteck um runde Objekte herum legen, wodurch die Zeit zum Berechnen aller Rechtecke dann natürlich wieder ansteigt.

Diese Optimierung hilft uns allerdings nur dann, wenn wir den Sehstrahl losschicken, um zu testen, ob wir überhaupt etwas sehen. Für den gespiegelten Sehstrahl oder für die Schattenberechnung nützt das alles nichts. Dies liegt daran, daß sich die Rechtecke am Bildschirm, also an der Projektionsebene orientieren, welcher nur für die Sichtbarkeitsanalyse relevant ist. Hier kommt uns entgegen, daß sowohl Bildschirm als auch Projektionspunkt fest sind, sich also während der Berechnung nicht ändern. Für den gespiegelten Sehstrahl können wir somit überhaupt keine Optimierung vornehmen, da weder der Ausgangspunkt des gespiegelten Sehstrahls noch dessen Richtung fest liegen.

Aber wie wär's mit dem Schatten. Die Lichtquelle verändert ihre Position ebenfalls nicht. Allerdings können die Oberflächen, die im Schatten liegen, überall sein, müssen sich also nicht auf eine Ebene beschränken, wie die Projektionsebene. Wenn wir jedoch statt der Rechtecke die Polarkoordinaten aller Objekte relativ zur Lichtquelle berechnen, und für jedes Objekt Minima und Maxima für die beiden Winkel festhalten, so können wir wenn Schatten! True ist und damit Schatten bestimmt werden soll, bei WelcherKoerper, abfragen, ob die Polarkoordinaten des Schnittpunktes innerhalb der beiden Intervalle liegen, die für jedes Objekt festgelegt sind. Puh!

Sie verstehen, um was es geht? Statt der kartesischen Koordinaten (x,y,z) können Sie einen Punkt auch in Polarkoordinaten angeben, und zwar als (a,b,d). Dabei sind a und b Winkel, und d ist der Abstand vom Koordinatenursprung. Für jedes Objekt können Sie nun ein Amin und ein Amax angeben, so daß für jeden Punkt des Objektes gilt, daß der Winkel a seiner Polarkoordinaten zwischen Amin und Amax liegt. Das gleiche können Sie für b tun, indem Sie ein Bmin und ein Bmax festlegen.

Wenn Sie dann bei WelcherKoerper feststellen wollen, ob ein Objekt einen Schatten auf einen Punkt wirft, so berechnen Sie zuerst die Polarkoordinaten des Punktes (pa,pb,pd). Bevor Sie die Schnittpunktsberechnung durchführen, testen Sie, ob pa zwischen Amin und Amax, und ob pb zwischen Bmin und Bmax für dieses Objekt liegt. Ist dies nicht der Fall, so kann dieses Objekt keinen Schatten auf den Punkt werfen, und Sie können sich die Schnittpunktsberechnung sparen.

Weiterhin können Sie noch den minimalen Abstand eines Objektes zur Lichtquelle festhalten. Dann können Sie zusätzlich noch testen, ob der Abstand des Objektes von der Lichtquelle nicht schon größer als der Abstand des Punktes von der Lichtquelle ist, so daß Sie sich auch dann die Schnittpunktsberechnung sparen können.

Hört sich kompliziert an und ist es auch. Es tritt dabei das Problem auf, daß der Winkel normalerweise zwischen -Pi und +Pi liegen kann. Liegt ein Objekt nun aber gerade auf dieser Trennungslinie zwischen +Pi und -Pi, so erhält man ein falsches Winkelintervall. In diesem Fall muß man auf den Winkel zuvor ein Inkrement addieren, so daß der Winkel danach zwischen 0 und 2*Pi liegt. Auch dieses Inkrement muß für jedes Objekt festgehalten, und bei der Abfrage berücksichtig werden. Das Feld, welches alle diese Werte beinhaltet, heißt MinMaxLq, und seine Elemente haben folgende Bedeutung:

```
MinMaxLq(n,0): Inkrement für Winkel a.
MinMaxLq(n,1): Minimum für Winkel a.
MinMaxLq(n,2): Maximum für Winkel a.
MinMaxLq(n,3): Inkrement für Winkel b.
MinMaxLq(n,4): Minimum für Winkel b.
MinMaxLq(n,5): Maximum für Winkel b.
MinMaxLq(n,6): Minimaler Abstand des Objektes n von Q.
```

Initialisiert wird dieses Feld mittels des Unterprogramms InitMinMaxLq. In WelcherKoerper sieht die Abfrage für MinMax und MinMaxLq folgendermaßen aus:

```
SUB WelcherKoerper (Kp%,Px,Py,Pz,Rx,Ry,Rz,Original!,Schatten!)
STATIC
SHARED True,False,minmax%(),AnzahlK,minmaxlq(),Hilf(),K(),xb%,yb%,
       Sx,Sy,Sz,Koerper%,Ac,Bc,Cc,la,Schwelle

  ' = > Koerper% = Nr des Körpers unter Koordinate xb%,yb%,
  ' S(sx,sy,sz)=Schnittpunkt, la=Geradenabschnitt
  ' = > falls Typ>=20:
      (ac,bc,cc) = transformierte Schnittpunktskoordinaten

  la=-1
  Koerper%=0

  IF Schatten! = True THEN

    'Polarkoordinaten des Schnittpunkts (Px,Py,Pz)
    'relativ zur Lichtquelle berechnen:

    CALL Calcablq(winkela,winkelb,0!,0!,Px,Py,Pz)
    Dist=SQR(Rx*Rx+Ry*Ry+Rz*Rz)
  END IF


  FOR n%=1 TO AnzahlK

    'Rechtecke abfragen:

    IF Original! = True THEN
      IF minmax%(n%,0)>xb% THEN
        GOTO Nxtk
      END IF
      IF minmax%(n%,1)>yb% THEN
        GOTO Nxtk
      END IF
      IF minmax%(n%,2)<xb% THEN
        GOTO Nxtk
      END IF
      IF minmax%(n%,3)<yb% THEN
        GOTO Nxtk
      END IF
    END IF

    'Winkelintervall abfragen:

    IF Schatten! = True THEN
```

```
        CALL Normwinkel(Wa,winkela+minmaxlq(n%,0))
        CALL Normwinkel(Wb,winkelb+minmaxlq(n%,3))
        IF Wa<minmaxlq(n%,1) THEN
          GOTO Nxtk
        END IF
        IF Wa>minmaxlq(n%,2) THEN
          GOTO Nxtk
        END IF
        IF Wb<minmaxlq(n%,4) THEN
          GOTO Nxtk
        END IF
        IF Wb>minmaxlq(n%,5) THEN
          GOTO Nxtk
        END IF
        IF Dist<minmaxlq(n%,6) THEN
          GOTO Nxtk
        END IF
      END IF

      'Hier kämen jetzt die Unterprogrammaufrufe für
      'die Schnittpunktsberechnung:

      ...

  Nxtk:
    NEXT n%

    IF Koerper%>0 AND Kp%=0 THEN
      Sx=Px+la*Rx
      Sy=Py+la*Ry
      Sz=Pz+la*Rz
    END IF

  END SUB
```

Ausgespart sind die Aufrufe für die Schnittpunktsberechnung, da Sie diese schon kennen. Ebenso ausgespart sind die Unterprogramme, welche in ihrer Summe die beiden Felder MinMax und MinMaxLq initialisieren, da diese fast noch mehr Platz einnehmen, als die Routinen, die für das eigentliche Ray-Tracing zuständig sind.

Lassen Sie mich zuletzt noch ein paar Worte zur Berechnungszeit verlieren. Vielleicht wissen Sie, daß selbst Superrechner wie eine Cray zur Berechnung eines Bildes gute zwanzig Minuten braucht. Zwar in höherer Auflösung und mit sehr vielen Objekten, aber trotzdem. Die Routinen, die Sie auf Ihrer Diskette finden, sind in BASIC geschrieben, hauptsächlich, weil BASIC eine einfach zu verstehende Sprache ist. Allerdings ist BASIC eine eher langsame Sprache. Sie ahnen schon, worauf ich hinaus will: Das Berechnen eines Bildes kostet Zeit, viel Zeit. Glücklicherweise ist der Amiga multitaskingfähig, so daß Sie während der Berechnung nicht unbedingt auf ihn verzichten müssen.

Empfehlenswert ist es aber auf jeden Fall, das Programm, das Sie sich aus den Routinen zusammenbauen können, compilieren zu lassen. Dann sinkt die Rechenzeit für ein 320x200 Punkte großes Bild auf unter einen Tag. Wenn Sie nur wenige Objekte definiert haben, die am besten nicht alle verspiegelt sind, so sinkt die Rechenzeit auf einige Stunden, womit sich durchaus leben läßt.

Vielleicht fallen Ihnen ja noch weitere Optimierungen ein, oder Sie schreiben das Programm in Assembler, was nach meinen überschlägigen Schätzungen eine Geschwindigkeitssteigerung um den Faktor drei bis vier bringen müßte gegenüber einer compilierten Version. Ganz fein sind natürlich alle Besitzer von Arithmetikcoprozessoren raus, die vermutlich nur noch ein paar Minuten warten müssen; allerdings fehlen mir hier Erfahrungswerte. Wenn Sie dagegen mit der AmigaBASIC-Version arbeiten wollen, so betrachten Sie dies als eines der letzten Abenteuer, die es auf diesem Planeten noch gibt.

4. Realisierung des Tracers

Nun wollen wir Ihnen zeigen, zu welchem Ergebnis wir gekommen sind. Die nun folgenden Kapitel enthalten unsere Realisierungsvorschläge für den Tracer. Für die Beschreibung der Routinen möchten wir aber hier schon festhalten: Es geht weniger darum zu zeigen, wie man programmiert, sondern mehr darum, zu zeigen, wie man dreidimensionale Grafiken programmiert.

4.1 Mehr dreidimensionale Grafik!

Nachdem Sie schon erfahren haben, wie das sogenannte Ray-Tracing, oder zu deutsch die Strahlrückverfolgung, theoretisch funktioniert, wollen wir Ihnen in den folgenden Kapiteln zunächst erklären, wie die dreidimensionalen Drahtmodelle erzeugt werden.

4.1.1 Von 3-D nach 2-D

Diese Drahtmodelle werden benötigt, um vor dem Schattieren, das sehr viel Zeit in Anspruch nehmen kann, einen ungefähren Eindruck von der Szene in der Computer-Welt zu vermitteln.

Dazu müssen wir alle dreidimensionalen Punkte, deren X-,Y- und Z-Koordinaten angegeben wurden, irgendwie auf den Bildschirm projizieren. Punkte aus dem Raum müssen also auf eine Ebene abgebildet werden. Diese auf eine Ebene projizierten Punkte können wir dann auf den Bildschirm ausgeben.

Aus dem Bild auf dem Bildschirm muß ersichtlich sein, ob ein Körper vor oder hinter einem anderen plaziert ist. Wir müssen also einen perspektivischen Eindruck erzeugen, der die im Computer enthaltene Szene möglichst realistisch wiedergibt.

Wir legen dazu zuerst unseren Standort innerhalb der Computer-Welt fest. Dies tun wir, indem wir die Koordinaten unseres Standpunktes in den Variablen 'Px','Py' und 'Pz' angeben. Den so bestimmten Punkt nennen wir 'Projektionspunkt'.

Wo schauen wir aber hin? Mit einem im Raum festgelegten Punkt können wir die Blickrichtung nicht ausmachen. Wir legen also einen zweiten Punkt fest, der (auch) für die Blickrichtung zuständig ist. Diesen Punkt nennen wir Hauptpunkt und bestimmen dessen Koordinaten in 'Hx','Hy' und 'Hz'.

Wir 'befinden' uns also am Projektionspunkt, und schauen in Richtung Hauptpunkt.

Jedoch ist das Festlegen der Blickrichtung nicht die einzige Aufgabe des Hauptpunktes. Dieser Punkt ist nämlich gleichzeitig ein Punkt unserer Projektionsebene - die Ebene, auf die unsere Punkte projiziert werden. Bevor die Punkte auf den Bildschirm ausgegeben werden, werden sie erst einmal auf die Projektionsebene abgebildet.

Aber auch der Projektionspunkt ist nicht ganz unbeteiligt. Die Gerade durch Projektionspunkt und Hauptpunkt steht nämlich senkrecht auf der Projektionsebene:

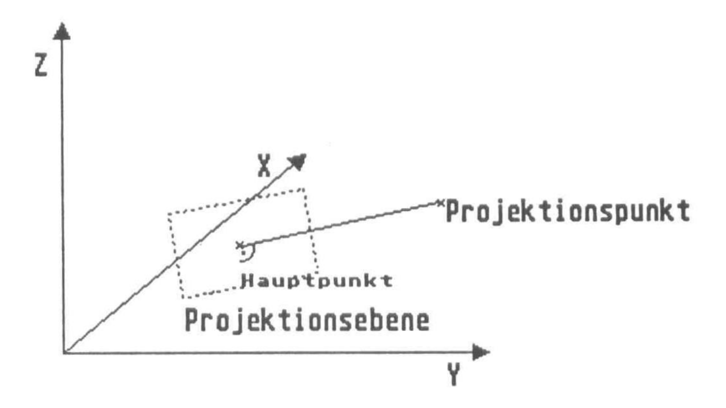


Abbildung 4.1

Diese Gerade PH (Projektionspunkt-Hauptpunkt) hilft uns nun, die Drehwinkel um die Koordinatenachsen zu bestimmen, um die die Projektionsebene gedreht werden muß, damit sie mit der YZ-Ebene zusammenfällt.

Dazu legen wir zuerst den Hauptpunkt in den Ursprung. Erreicht wird dies dadurch, daß von den Koordinaten aller Punkte die Koordinaten des Hauptpunktes abgezogen werden.

Wie wir die Drehwinkel bestimmen können, wird aus folgender Abbildung ersichtlich:

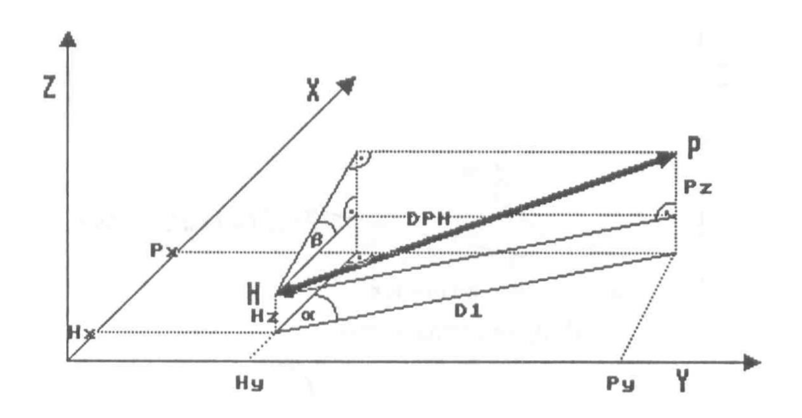


Abbildung 4.2

Wie Sie sehen, gelten für den Winkel Alpha (Z-Achse) und den Winkel Beta (Y-Achse) die Beziehungen

```
Sin(α) = (Px-Hy)/D1
Cos(α) = (Px-Hx)/D1

Sin(ß) = (Pz-Hz)/DPH   (DPH = Distanz Projektions-Hauptpunkt)
Cos(ß) = D1/DPH.
```

Die eigentlichen Winkel kann man dann mittels der 'Arcus Tangens'-Funktion des AmigaBASIC berechnen:

```
Winkel = ATN(Sin/Cos),
```

da der Tangens eines Winkels auch als Quotient aus Sinus und Cosinus des entsprechenden Winkels definiert wird.

Den Winkel um die X-Achse müssen wir aber zunächst gleich 0° setzen. Denn wie aus der Abbildung ersichtlich, läßt sich dieser Drehwinkel, nachdem die Projektionsebene in der YZ-Ebene liegt, bzw. der Projektionspunkt auf der X-Achse liegt, nicht mehr errechnen.

Die Routine 'InitialP' errechnet uns aus der Position des Hauptpunktes und des Projektionspunktes die benötigten Drehwinkel, sowie die Distanz vom Hauptpunkt zum Projektionspunkt.

```
InitialP:
  D1=SQR((Px-Hx)^2+(Py-Hy)^2)
  DPH=SQR((Px-Hx)^2+(Py-Hy)^2+(Pz-Hz)^2)

  IF D1=0 THEN
    Sina=0
    Cosa=1
  ELSE
    Sina=(Py-Hy)/D1
    Cosa=(Px-Hx)/D1
  END IF

  IF DPH=0 THEN
    Sinb=0
    Cosb=1
  ELSE
    Sinb=(Pz-Hz)/DPH
    Cosb=D1/DPH
  END IF

  Sinc=0                          ' Winkel um Z-Achse nicht
  Cosc=1                          ' eindeutig bestimmbar

  IF Cosa=0 THEN
    Alpha=Pd2
  ELSE
    Alpha=ATN(Sina/Cosa)            ' aus Cosinus und Sinus
  END IF                            ' Winkel berechnen
```

```
    IF Cosa<0 THEN
      Alpha = Alpha+Pi
    END IF

    IF Cosb=0 THEN
      Beta=Pd2
    ELSE
      Beta=ATN(Sinb/Cosb)
    END IF

    IF Cosb<0 THEN
      Beta = Beta + Pi
    END IF

    Gamma=0
  RETURN
```

Wenn wir nun also die Drehwinkel berechnet haben, brauchen wir nur noch dafür zu sorgen, daß die Punkte um die Koordinatenachsen gedreht werden. Wir müssen die einzelnen Punkte denselben Drehungen unterziehen wie die Projektionsebene, damit diese auf der YZ-Ebene liegt.

Wie dreht man aber einen Punkt?

Betrachten wir dazu diese Abbildung:

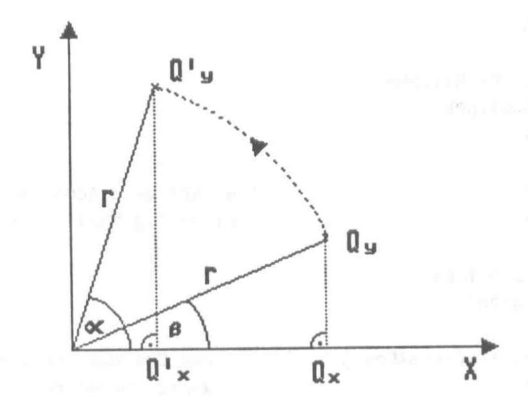


Abbildung 4.3

Der Punkt Q wird um den Winkel α gedreht. Q' ist der um den Winkel Alpha gedrehte Punkt. Es gelten nun aber folgende Beziehungen:

```
Sin(ß) = Qy/r          Sin(α-ß) = Q'y/r
Cos(ß) = Qx/r          Cos(α-ß) = Q'x/r
```

Setzt man dies in die Additionstheoreme ein, erhält man die Formel für das Drehen eines Punktes um den Winkel Alpha.

Additionstheoreme:

```
Cos(ß-α) = Cos(ß)*Cos(α)+Sin(ß)*Sin(α)
Sin(ß-α) = Sin(ß)*Cos(α)-Cos(ß)*Sin(α)
```

Einsetzungen:

```
Q'x/r = Cos(α)*Qx*r + Sin(α)*Qy*r
Q'y/r = Cos(α)*Qy*r - Sin(α)*Qx*r

  Q'x = Cos(α)*Qx + Sin(α)*Qy
  Q'y = Cos(α)*Qy - Sin(α)*Qx
```

Nun brauchen wir die Punkte nur noch nacheinander nach obiger Formel um alle drei Koordinaten-Achsen zu drehen. Drehen wir einen Punkt z.B. um die Z-Achse, so wird die Z-Koordinate des gedrehten Punkts nicht von der Drehung betroffen - sie bleibt gleich. Ebenso ist es bei Drehungen um die anderen beiden Achsen. Die jeweilige Koordinate wird nicht verändert.

Die auszuführenden Drehungen sind in folgender Abbildung exemplarisch durchgefürt worden:

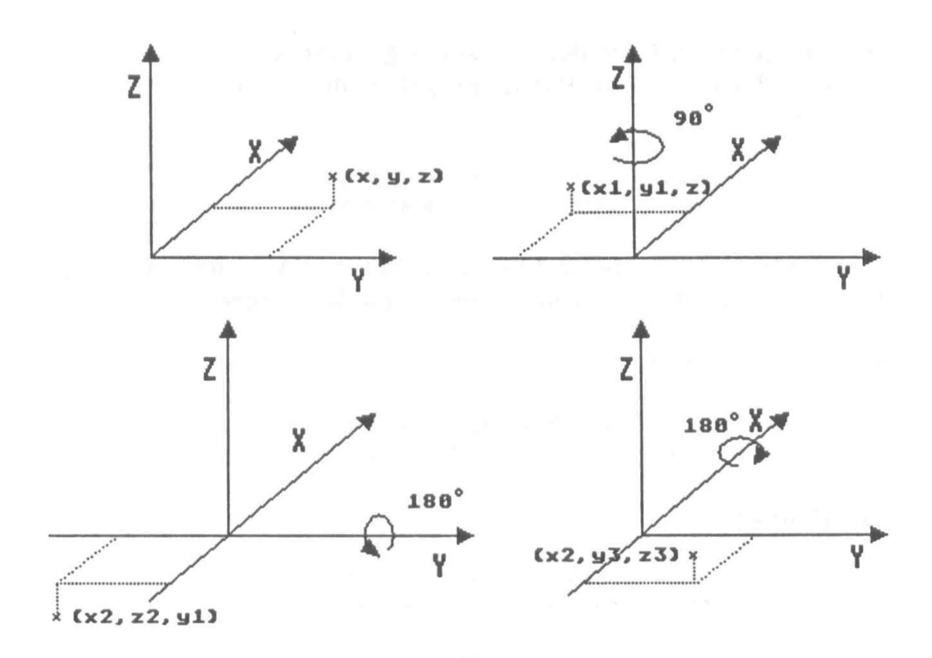
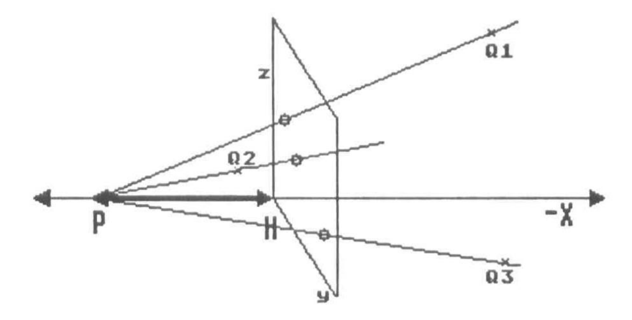


Abbildung 4.4

Beachten sollten Sie, daß für die Drehung um die Y-Achse schon berechnete Koordinaten der Drehung um die Z-Achse benutzt wurden. Bei der Drehung um die X-Achse wurden Ergebnisse der Y-Drehung verwendet.

Jetzt können wir uns der Perspektive widmen. Dazu betrachten wir die Geraden durch den Projektionspunkt und die schon um die Koordinatenachsen gedrehten Punkte:



Projektionsebene

Abbildung 4.5

Wie Sie sehen, hat der Projektionspunkt nun die Koordinaten (DPH,0,0), was ja auch ganz verständlich ist. Aus der Geraden Projektions-Hauptpunkt haben wir die Drehwinkel um die Z- und Y-Achse bestimmt, damit die Projektionsebene auf der YZ-Ebene liegt. Da die Gerade PH aber senkrecht auf dieser Ebene liegt, und weiterhin der Hauptpunkt in den Koordinatenursprung verlegt wurde, bedeutet das, daß der Projektionspunkt nach dieser Drehung in der Entfernung DPH auf der X-Achse liegt.

Doch betrachten wir nun die Schnittpunkte der Geraden mit der Ebene. Jeder dieser Schnittpunkte stellt einen darzustellenden Punkt dar. Die Koordinaten der einzelnen Schnittpunkte können wir also für unsere Bildschirmausgabe benutzen.

Wir aber bestimmt man diese Schnittpunkte?

Sehen wir uns dazu die Geradengleichung der Gerade Projektionspunkt - darzustellender Punkt an:

$$\begin{matrix} X \\ Y \\ Z \end{matrix} = \begin{matrix} Qx \\ Qy \\ Qz \end{matrix} + t * \begin{matrix} (Px-Qx) \\ (Py-Qy) \\ (Pz-Qz) \end{matrix}$$

Wir wissen jedoch, daß der Schnittpunkt zwangsläufig die X-Koordinate 0 haben muß, denn die X-Koordinate aller Punkte der YZ-Ebene ist gleich 0 (Wir haben ja durch die Drehungen erreicht, daß die Projektionsebene in der YZ-Ebene liegt).

Zusammen mit den Koordinaten des Projektionspunktes und der X-Koordinate des Schnittpunktes können wir folgende Beziehung aufstellen:

```
X   Qx           (DPH-Qx)
Y = Qy + t *  (0-Qy)
Z   Qz           (0-Qz)

X = Qx + t * (DPH-Qx) = 0
                    t = -Qx/(DPH-Qx)
```

und durch Einsetzen von 't' und Umformen der beiden verbleibenden Gleichungen erhält man für die Y- und Z-Koordinate des Schnittpunktes: (Q'ist der schon um alle 3 Achsen gedrehte Punkt):

```
Y = Q'y*DPH/(DPH-Qx)
Z = Q'z*DPH/(DPH-Qx)
```

Dies sind unsere Bildschirmkoordinaten. Allerdings müssen wir nun ein wenig umdenken. Wir müssen nämlich die Z-Koordinate des Schnittpunktes als eigentliche Y-Bildschirmkoordinate und die Y-Koordinate des Schnittpunktes als eigentliche X-Koordinate des Bildschirms betrachten.

Die nun folgende Routine übernimmt die gesamte 3-D/2-D-Transformation, also das Drehen um die Koordinatenachsen und die Perspektivtransformation - die zurückgegebenen Punkte müssen Sie dann nur noch auf den Bildschirm ausgeben.

```
SUB Projektion(Px%,Py%,x,y,z) STATIC
SHARED Sina,Sinb,Sinc,Cosa,Cosb,Cosc,DPH,Hx,Hy,Hz
  ' 3-D Koordinaten (x,y,z) nach 2-D (Px%,Py%) = Bildschirm
```

```
    x1=(x-Hx)*Cosa+(y-Hy)*Sina  ' Drehung Z-Achse
    y1=(y-Hy)*Cosa-(x-Hx)*Sina
    x2=x1*Cosb+(z-Hz)*Sinb      ' Drehung Y-Achse
    z2=(z-Hz)*Cosb-x1*Sinb
    y3=y1*Cosc+z2*Sinc          ' Drehung X-Achse
    z3=z2*Cosc-y1*Sinc

    IF DPH<>x3 THEN
      Px%=FN Xscale((y3*DPH)/(DPH-x2)) -4 ' Schnittpunkt mit
                                          ' Projektionsebene
      Py%=FN Yscale((z3*DPH)/(DPH-x2)) -2 ' skalieren
    ELSE
      Px% = FN Xscale(0) -4
                    ' Subtraktion von 4 und 2 aufgrund des Zeichnens
      Py% = FN Yscale(0) -2               ' in ein GIMMEZEROZERO-
                                          ' Window (BASIC-Window)
    END IF
  END SUB
```

Bei eingehender Betrachtung dieser Routine sind Ihnen aber sicherlich folgende Funktionen ins Auge gefallen:

```
FN XScale (x) = (x+BildX) * Bildbreite    und
FN YScale (y) = (BildY-y) * Bildhoehe
```

Sie sorgen dafür, daß die Schnittpunkte mit der YZ-Ebene in der richtigen Vergrößerung dargestellt werden. Um jedoch zu verstehen, was diese Benutzerfunktionen bewirken, müssen erst einmal die Bedeutungen der Variablen klar sein:

'BildBreite' und 'BildHoehe' sind die Vergrößerungsfaktoren in X- und Y-Richtung. 'BildX' und 'BildY' geben dagegen die Größe des Ausschnittfensters an, dessen Inhalt vergrößert werden soll:

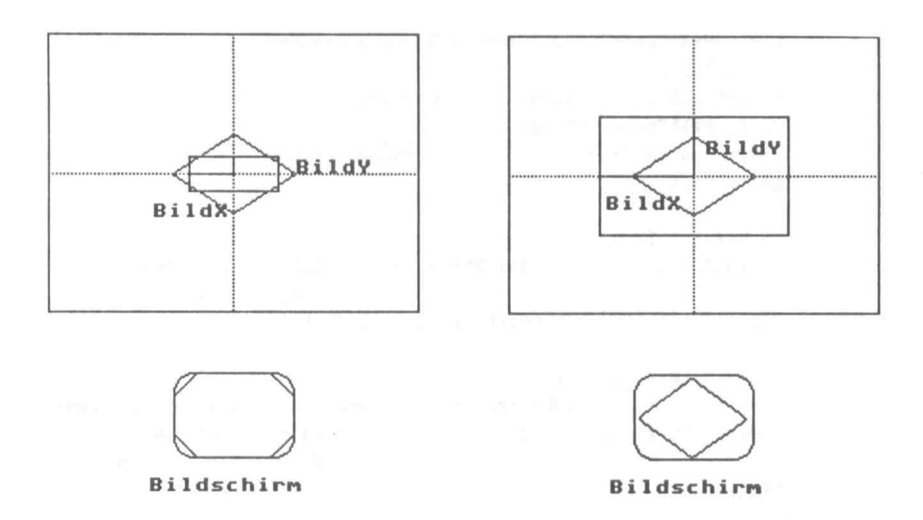


Abbildung 4.6

Je größer 'BildX' und 'BildY', desto größer ist der Bereich der Punkte, die auf dem Bildschirm dargestellt werden müssen. Berechnet werden 'BildX' und 'BildY' so:

```
BildX = RasterW2% / BildBreite
BildY = RasterH2% / BildHoehe
```

RasterW2% und RasterH2% sind dabei die Bildschirmkoordinaten des Mittelpunktes (Bei einer Auflösung von 320x200 Punkten wäre RasterW2% = 160 und RasterH2% = 100).

Hieraus wird ersichtlich, daß 'BildX' und 'BildY' immer größer werden, wenn 'BildBreite' und 'BildHoehe' immer kleiner werden, und umgekehrt.

'XScale' und 'YScale' sorgen nun dafür, daß der Ursprung des Koordinatensystems (= transformierter Hauptpunkt) an den Mittelpunkt des Bildschirms transformiert wird.

Dies wird dadurch erreicht, daß zu 'BildX' die X-Koordinate des Schnittpunktes addiert wird. Von der Y-Koordinate des

Schnittpunktes zieht man 'BildY' ab, weil die Y-Koordinaten von oben nach unten, und nicht wie vom kartesischen Koordinatensystem her gewohnt von unten nach oben verlaufen.

Die so gewonnen Koordinaten multipliziert man dann noch einmal mit den Vergrößerungsfaktoren, um die Punkte bildschirmgerecht zu machen.

Bis jetzt waren wir immer davon ausgegangen, daß ein Hauptpunkt und ein Projektionspunkt vorgegeben worden waren. Grundsätzlich reichen diese beiden Punkte aus, um eine voll funktionsfähige 3-D/2-D-Routine zu entwickeln.

Allerdings muß man diese Punkte auch im Programm ändern können. Muß man nämlich erst ein Programm unterbrechen, und eine oder mehrere Programmzeilen ändern, um z.B. neue Koordinaten des Hauptpunktes zu bestimmen, wird das auf die Dauer sehr lästig.

Deshalb können Sie in folgenden Routinen den Hauptpunkt ändern, einen neuen Projektionspunkt eingeben, die Drehwinkel nachträglich ändern (in Grad und nicht in Radiant, wie man aufgrund des AmigaBASIC vermuten könnte) und den Abstand DPH vergößern oder verkleinern (je kleiner DPH wird, desto verzerrter ist die dreidimensionale Darstellung auf dem Bildschirm. Wird DPH jedoch sehr groß gewählt, geht der zentralperspektivische Eindruck verloren - das Bild wird (fast) parallelperspektivisch).

```
InputH:
  Status$ = " Status: Hauptpunkt"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  LOCATE 10,1
  PRINT "         Hauptpunkt: "

  LOCATE 12,1
  PRINT "     Hx = ";
  CALL FormInput (Hx,30!,-1E+14,1E+14)
```

```
    LOCATE 13,1
    PRINT "     Hy = ";
    CALL FormInput (Hy,30!,-1E+14,1E+14)

    LOCATE 14,1
    PRINT "     Hz = ";
    CALL FormInput (Hz,30!,-1E+14,1E+14)

    GOSUB Initial
    Neu! = True
    CLS
    GOSUB MakeMenu
  RETURN


  InputP:
    Status$ = " Status: Projektionspunkt"+CHR$(0)
    CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

    GOSUB DeleteMenu

    LOCATE 10,1
    PRINT "           Projektionspunkt: "

    LOCATE 12,1
    PRINT "           Px = ";
    CALL FormInput (Px,30!,-1E+14,1E+14)

    LOCATE 13,1
    PRINT "           Py = ";
    CALL FormInput (Py,30!,-1E+14,1E+14)

    LOCATE 14,1
    PRINT "           Pz = ";
    CALL FormInput (Pz,30!,-1E+14,1E+14)

    GOSUB InitialP
    Neu! = True
    CLS
    GOSUB MakeMenu
  RETURN
```

```
InputWinkel:
  Status$ = " Status: Drehwinkel eingeben"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  a=FN Deg(Alpha)
  b=FN Deg(Beta)
  c=FN Deg(Gamma)

  LOCATE 10,1
  PRINT "         Drehwinkel (a,■,c): "

  LOCATE 12,1
  PRINT "     Alpha (Z-Achse) = ";
  CALL FormInput (a,30!,0!,360!)

  LOCATE 13,1
  PRINT "     Beta  (Y-Achse) = ";
  CALL FormInput (b,30!,0!,360!)

  LOCATE 14,1
  PRINT "     Gamma (X-Achse) = ";
  CALL FormInput (c,30!,0!,360!)

  Alpha=FN Rad(a)
  Beta=FN Rad(b)
  Gamma=FN Rad(c)

  GOSUB Initial
  Neu! = True
  CLS
  GOSUB MakeMenu
RETURN


InputDPH:
  Status$ = " Status: Abstand"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  LOCATE 10,1
  PRINT "     Abstand zur Projektionsfläche = ";
  CALL FormInput (DPH,30!,-1E+14,1E+14)
```

```
    GOSUB Initial
    Neu! = True
    CLS
    GOSUB MakeMenu
  RETURN
```

Bei der Änderung des Hauptpunktes, des Abstandes DPH oder der Drehwinkel, sollte man beachten, daß dann die Position des Projektionspunktes verändert wird. Der Hauptpunkt wird hierzu als 'Basispunkt' herangezogen.

In 'Initial' wird aufgrund dieser Daten der neue Projektionspunkt berechnet:

```
Initial:
  WHILE Alpha<0
     Alpha = Alpha + Pm2
  WEND
  WHILE Beta<0
     Beta = Beta + Pm2
  WEND
  WHILE Gamma<0
     Gamma = Gamma + Pm2
  WEND
  WHILE Alpha>Pm2
     Alpha = Alpha - Pm2
  WEND
  WHILE Beta>Pm2
     Beta = Beta - Pm2
  WEND
  WHILE Gamma>Pm2
     Gamma = Gamma - Pm2
  WEND
  Sina=SIN(Alpha)
  Cosa=COS(Alpha)
  Sinb=SIN(Beta)
  Cosb=COS(Beta)
  Sinc=SIN(Gamma)
  Cosc=COS(Gamma)
```

```
    Px=Hx+DPH*Cosa*Cosb   ' Projektionspunkt aufgrund
    Py=Hy+DPH*Sina*Cosb   ' des Hauptpunkts und des
    Pz=Hz+DPH*Sinb        ' Abstandes DPH und Alpha, Beta und Gamma
  RETURN
```

So können wir uns also mit dem Projektionspunkt, der ja unseren Standpunkt angibt, um den Hauptpunkt 'herum bewegen'.

4.1.2 Körper und Flächen

Nachdem wir nun in der Lage sind, Punkte aus dem Raum auf den Bildschirm auszugeben, wollen wir uns nun damit beschäftigen, wie man Flächen und Körper darstellt.

Das Prinzip, das dahinter steckt, ist dabei sehr einfach: Aus gegebenen Informationen über die Körper und Flächen berechnet man einige markante Punkte, projiziert diese auf den Bildschirm und verbindet die Punkte auf dem Bildschirm mit Linien.

Die Daten über die einzelnen Körper und Flächen erhalten wir aus dem Array K(), das mit 'Dim K(AnzahlK,5,2)' dimensioniert wurde. Wie aber die einzelnen Array-Elemente aussehen müssen, bzw. welche Werte diese enthalten müssen, um einen bestimmten Körper oder eine bestimmte Fläche darzustellen, können Sie den nun folgenden Abbildungen entnehmen:

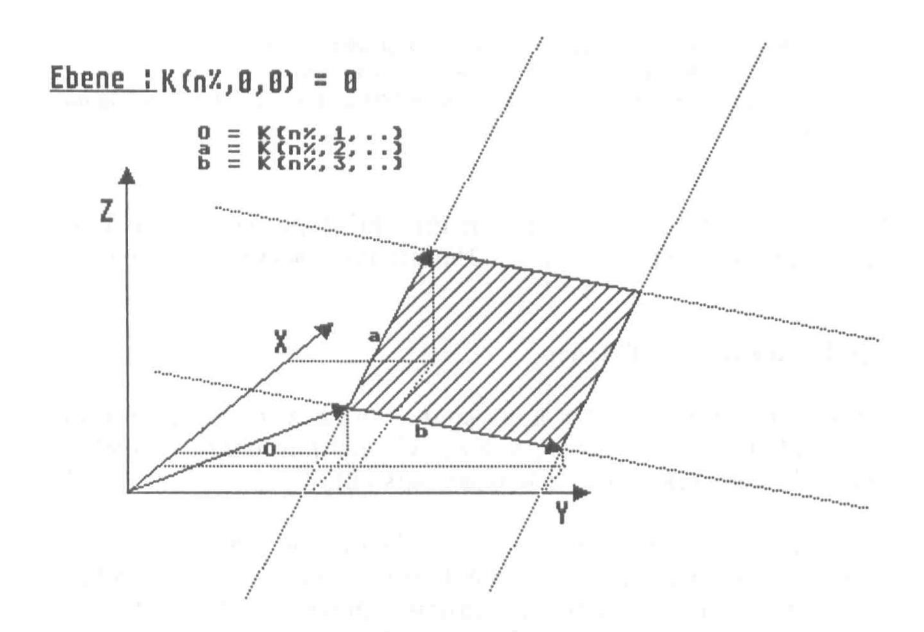


Abbildung 4.7

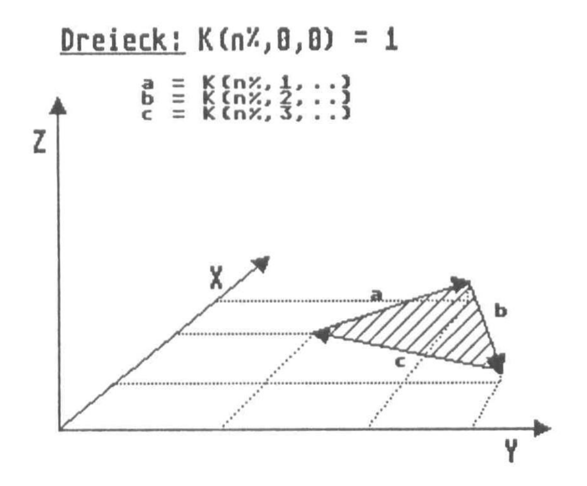


Abbildung 4.8

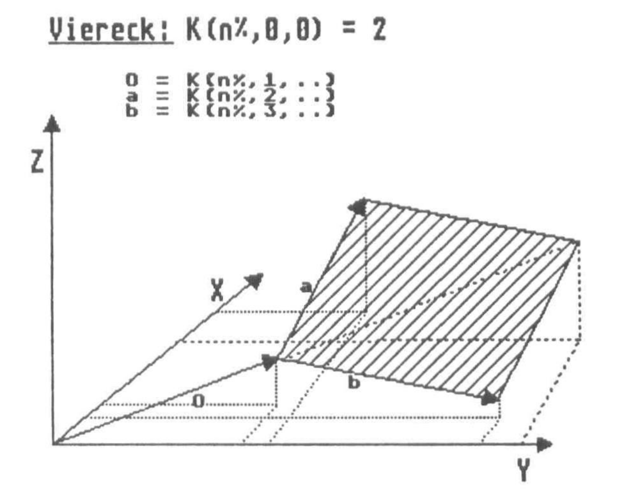
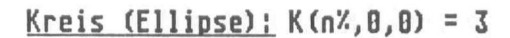


Abbildung 4.9



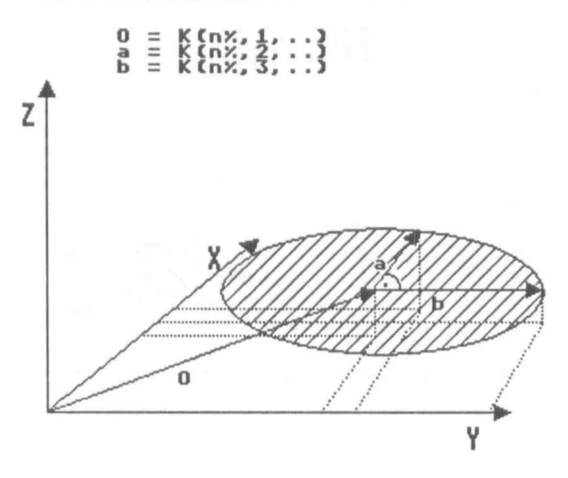


Abbildung 4.10

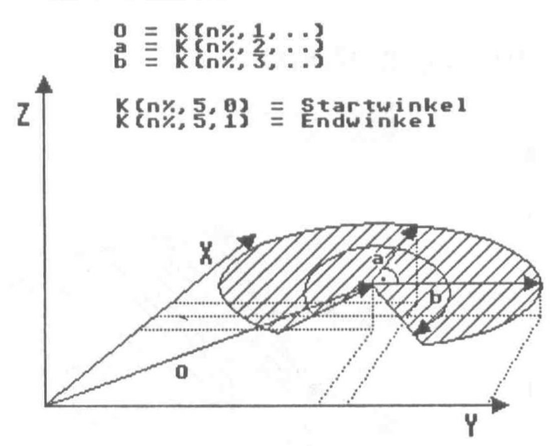


Abbildung 4.11

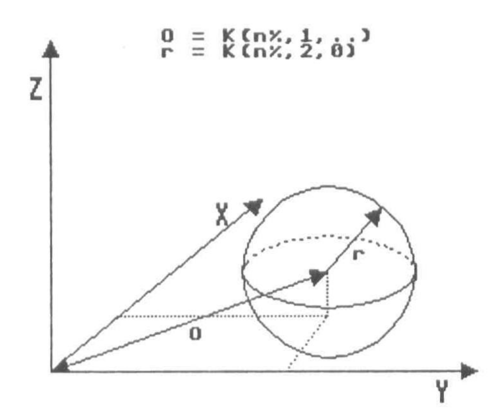


Abbildung 4.12

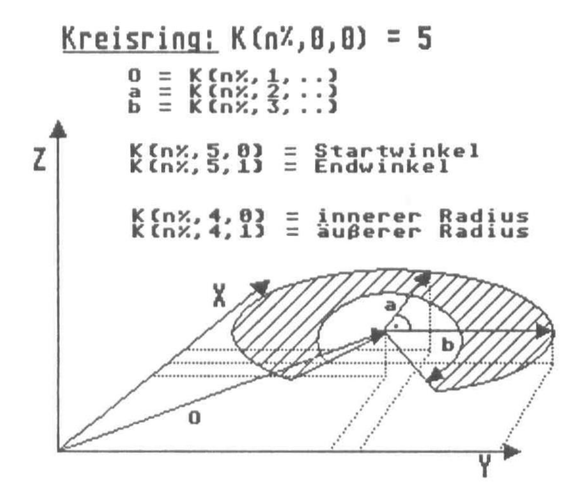


Abbildung 4.13

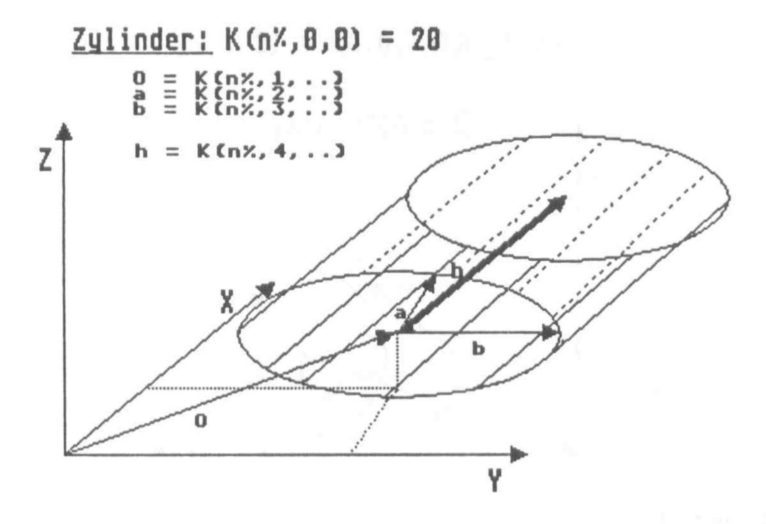


Abbildung 4.14



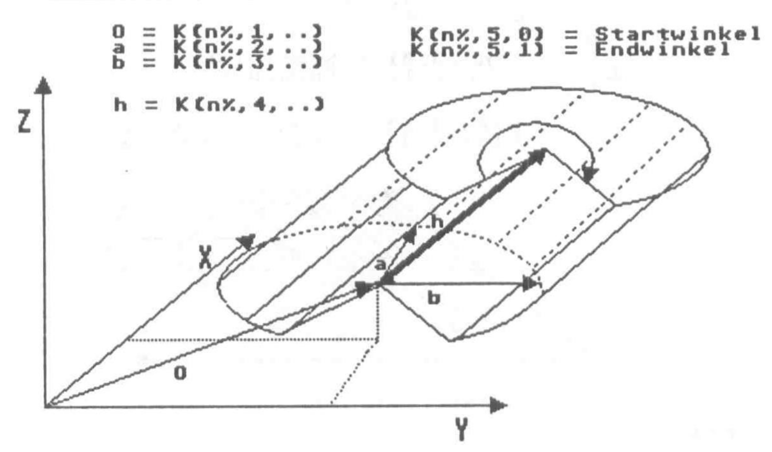


Abbildung 4.15

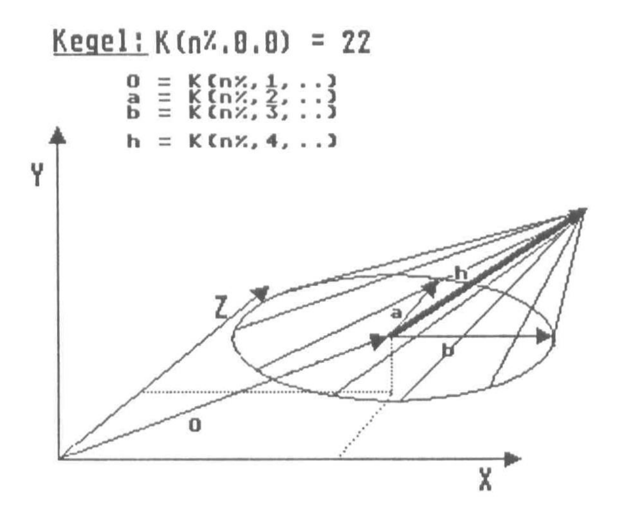


Abbildung 4.16

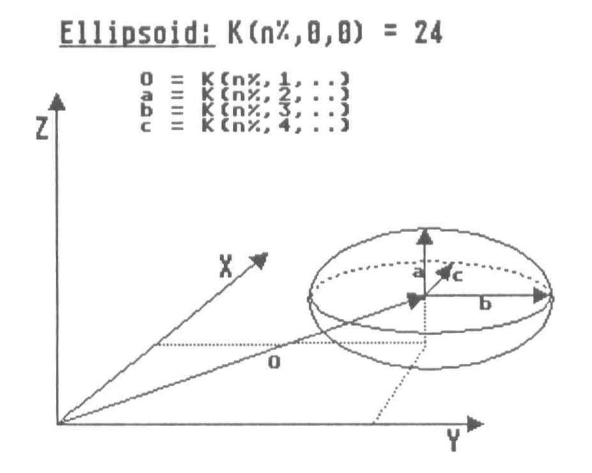


Abbildung 4.17

Die folgenden Routinen übernehmen dabei das Zeichnen der einzelnen Körper und Flächen:

```
ZeichneEbene:
  CALL
Projektion(Xp%,Yp%,K(n%,2,0)+K(n%,1,0),K(n%,2,1)+K(n%,1,1),K(n%,2,2)+
K(n%,1,2))
  CALL Move&(RastPort&,Xp%,Yp%)
  CALL Projektion(Xp%,Yp%,K(n%,1,0),K(n%,1,1),K(n%,1,2))
  CALL Draw&(RastPort&,Xp%,Yp%)
  CALL
Projektion(Xp%,Yp%,K(n%,3,0)+K(n%,1,0),K(n%,3,1)+K(n%,1,1),K(n%,3,2)+
K(n%,1,2))
  CALL Draw&(RastPort&,Xp%,Yp%)
RETURN
'
ZeichneDreieck:
  CALL Projektion(x1%,y1%,K(n%,1,0),K(n%,1,1),K(n%,1,2))
  CALL Move&(RastPort&,x1%,y1%)
  CALL
Projektion(Xp%,Yp%,K(n%,2,0)+K(n%,1,0),K(n%,2,1)+K(n%,1,1),K(n%,2,2)+
K(n%,1,2))
  CALL Draw&(RastPort&,Xp%,Yp%)
  CALL
Projektion(Xp%,Yp%,K(n%,3,0)+K(n%,1,0),K(n%,3,1)+K(n%,1,1),K(n%,3,2)+
K(n%,1,2))
  CALL Draw&(RastPort&,Xp%,Yp%)
  CALL Draw&(RastPort&,x1%,y1%)
RETURN
'
ZeichneViereck:
  CALL Projektion(x1%,y1%,K(n%,1,0),K(n%,1,1),K(n%,1,2))
  CALL Move&(RastPort&,x1%,y1%)
  CALL
Projektion(Xp%,Yp%,K(n%,2,0)+K(n%,1,0),K(n%,2,1)+K(n%,1,1),K(n%,2,2)+
K(n%,1,2))
  CALL Draw&(RastPort&,Xp%,Yp%)
  Dx=K(n%,2,0)+K(n%,3,0)+K(n%,1,0)
  Dy=K(n%,2,1)+K(n%,3,1)+K(n%,1,1)
  Dz=K(n%,2,2)+K(n%,3,2)+K(n%,1,2)
  CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
  CALL Draw&(RastPort&,Xp%,Yp%)
  CALL
Projektion(Xp%,Yp%,K(n%,3,0)+K(n%,1,0),K(n%,3,1)+K(n%,1,1),K(n%,3,2)+
K(n%,1,2))
  CALL Draw&(RastPort&,Xp%,Yp%)
  CALL Draw&(RastPort&,x1%,y1%)
RETURN
```

```
'
ZeichneKreis:
  CALL
Projektion(Xp%,Yp%,K(n%,1,0)+K(n%,2,0),K(n%,1,1)+K(n%,2,1),K(n%,1,2)+
K(n%,2,2))
' da SIN(0) = 0 und COS(0) = 1 fällt K(n%,3,..) weg, und K(n%,2,..) wird
übernommen
  CALL Move&(RastPort&,Xp%,Yp%)
  w=Pm2/AnzahlSegmente
  D=w
  Repeat1:
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Draw&(RastPort&,Xp%,Yp%)
    w = w+D
  IF (w<=Pm2+D/2) THEN GOTO Repeat1:
RETURN
'
ZeichneKreisSektor:
  CALL Projektion(x1%,y1%,K(n%,1,0),K(n%,1,1),K(n%,1,2))
  w=K(n%,5,0)
  Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)
  Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)
  Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)
  CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
  CALL Move&(RastPort&,x1%,y1%)
  CALL Draw&(RastPort&,Xp%,Yp%)
  D=Pm2/AnzahlSegmente
  WHILE w<K(n%,5,1)
    w = w+D
    IF w>K(n%,5,1) THEN
      w=K(n%,5,1)
    END IF
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Draw&(RastPort&,Xp%,Yp%)
  WEND
  CALL Draw&(RastPort&,x1%,y1%)
RETURN
'
```

```
ZeichneKreisRing:
  w=K(n%,5,0)
  Dx=K(n%,1,0)+(K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w))*K(n%,4,0)
  Dy=K(n%,1,1)+(K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w))*K(n%,4,0)
  Dz=K(n%,1,2)+(K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w))*K(n%,4,0)
  CALL Projektion(Xua%,Yua%,Dx,Dy,Dz)
  Dx=K(n%,1,0)+(K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w))*K(n%,4,1)
  Dy=K(n%,1,1)+(K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w))*K(n%,4,1)
  Dz=K(n%,1,2)+(K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w))*K(n%,4,1)
  CALL Projektion(Xoa%,Yoa%,Dx,Dy,Dz)
  CALL Move&(RastPort&,Xua%,Yua%)
  CALL Draw&(RastPort&,Xoa%,Yoa%)
  D=Pm2/AnzahlSegmente
  WHILE w<K(n%,5,1)
    w = w+D
    IF w>K(n%,5,1) THEN
      w=K(n%,5,1)
    END IF
    Dx=K(n%,1,0)+(K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w))*K(n%,4,0)
    Dy=K(n%,1,1)+(K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w))*K(n%,4,0)
    Dz=K(n%,1,2)+(K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w))*K(n%,4,0)
    CALL Projektion(Xu%,Yu%,Dx,Dy,Dz)
    Dx=K(n%,1,0)+(K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w))*K(n%,4,1)
    Dy=K(n%,1,1)+(K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w))*K(n%,4,1)
    Dz=K(n%,1,2)+(K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w))*K(n%,4,1)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Move&(RastPort&,Xua%,Yua%)
    CALL Draw&(RastPort&,Xu%,Yu%)
    CALL Move&(RastPort&,Xp%,Yp%)
    CALL Draw&(RastPort&,Xoa%,Yoa%)
    Xua%=Xu%
    Yua%=Yu%
    Xoa%=Xp%
    Yoa%=Yp%
  WEND
  CALL Move&(RastPort&,Xua%,Yua%)
  CALL Draw&(RastPort&,Xoa%,Yoa%)
RETURN
'
ZeichneKugel:
  D=Pm2/AnzahlSegmente
  FOR w1=-Pd2+D TO Pd2-D/2 STEP D
    Dx=K(n%,1,0)
    Dy=K(n%,1,1)+K(n%,2,0)*COS(w1)
    Dz=K(n%,1,2)+K(n%,2,0)*SIN(w1)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
```

```
    CALL Move&(RastPort&,Xp%,Yp%)
    FOR w2=D TO Pm2+D/2 STEP D
      Dx=K(n%,1,0)+K(n%,2,0)*SIN(w2)*COS(w1)
      Dy=K(n%,1,1)+K(n%,2,0)*COS(w2)*COS(w1)
      Dz=K(n%,1,2)+K(n%,2,0)*SIN(w1)
      CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
      CALL Draw&(RastPort&,Xp%,Yp%)
    NEXT w2
  NEXT w1
  FOR w1=-Pd2+D TO Pd2-D/2 STEP D
    Dy=K(n%,1,1)
    Dz=K(n%,1,2)+K(n%,2,0)*COS(w1)
    Dx=K(n%,1,0)+K(n%,2,0)*SIN(w1)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Move&(RastPort&,Xp%,Yp%)
    FOR w2=D TO Pm2+D/2 STEP D
      Dy=K(n%,1,1)+K(n%,2,0)*SIN(w2)*COS(w1)
      Dz=K(n%,1,2)+K(n%,2,0)*COS(w2)*COS(w1)
      Dx=K(n%,1,0)+K(n%,2,0)*SIN(w1)
      CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
      CALL Draw&(RastPort&,Xp%,Yp%)
    NEXT w2
  NEXT w1
RETURN
'
ZeichneZylinder:
  Dx=K(n%,1,0)+K(n%,2,0)
  Dy=K(n%,1,1)+K(n%,2,1)
  Dz=K(n%,1,2)+K(n%,2,2)
  CALL Projektion(Xua%,Yua%,Dx,Dy,Dz)
  Dx=K(n%,1,0)+K(n%,2,0)+K(n%,4,0)
  Dy=K(n%,1,1)+K(n%,2,1)+K(n%,4,1)
  Dz=K(n%,1,2)+K(n%,2,2)+K(n%,4,2)
  CALL Projektion(Xoa%,Yoa%,Dx,Dy,Dz)
  D=Pm2/AnzahlSegmente
  FOR w=D TO Pm2+D/2 STEP D
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)
    CALL Projektion(Xu%,Yu%,Dx,Dy,Dz)
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)+K(n%,4,0)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)+K(n%,4,1)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)+K(n%,4,2)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Move&(RastPort&,Xua%,Yua%)
    CALL Draw&(RastPort&,Xu%,Yu%)
```

```
    CALL Draw&(RastPort&,Xp%,Yp%)
    CALL Draw&(RastPort&,Xoa%,Yoa%)
    Xua%=Xu%
    Yua%=Yu%
    Xoa%=Xp%
    Yoa%=Yp%
  NEXT w
RETURN
'
ZeichneZylinderSegm:
  w=K(n%,5,0)
  Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)
  Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)
  Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)
  CALL Projektion(Xua%,Yua%,Dx,Dy,Dz)
  Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)+K(n%,4,0)
  Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)+K(n%,4,1)
  Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)+K(n%,4,2)
  CALL Projektion(Xoa%,Yoa%,Dx,Dy,Dz)
  CALL Move&(RastPort&,Xua%,Yua%)
  CALL Draw&(RastPort&,Xoa%,Yoa%)
  D=Pm2/AnzahlSegmente
  WHILE w<K(n%,5,1)
    w = w+D
    IF w>K(n%,5,1) THEN
      w=K(n%,5,1)
    END IF
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)
    CALL Projektion(Xu%,Yu%,Dx,Dy,Dz)
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)+K(n%,4,0)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)+K(n%,4,1)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)+K(n%,4,2)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Move&(RastPort&,Xua%,Yua%)
    CALL Draw&(RastPort&,Xu%,Yu%)
    CALL Draw&(RastPort&,Xp%,Yp%)
    CALL Draw&(RastPort&,Xoa%,Yoa%)
    Xua%=Xu%
    Yua%=Yu%
    Xoa%=Xp%
    Yoa%=Yp%
  WEND
RETURN
```

```
'
ZeichneKegel:
  CALL
Projektion(x1%,y1%,K(n%,1,0)+K(n%,4,0),K(n%,1,1)+K(n%,4,1),K(n%,1,2)+
K(n%,4,2))
  CALL
Projektion(Xp%,Yp%,K(n%,1,0)+K(n%,2,0),K(n%,1,1)+K(n%,2,1),K(n%,1,2)+
K(n%,2,2))
  CALL Move&(RastPort&,Xp%,Yp%)
  D=Pm2/AnzahlSegmente
  FOR w=D TO Pm2+D/2 STEP D
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w)+K(n%,3,0)*SIN(w)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w)+K(n%,3,1)*SIN(w)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w)+K(n%,3,2)*SIN(w)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Draw&(RastPort&,Xp%,Yp%)
    CALL Move&(RastPort&,x1%,y1%)
    CALL Draw&(RastPort&,Xp%,Yp%)
  NEXT w
RETURN
'
ZeichneEllipsoid:
  D=Pm2/AnzahlSegmente
  FOR w1=-Pd2+D TO Pd2-D/2 STEP D
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w1)+K(n%,4,0)*SIN(w1)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w1)+K(n%,4,1)*SIN(w1)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w1)+K(n%,4,2)*SIN(w1)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Move&(RastPort&,Xp%,Yp%)
    FOR w2=D TO Pm2+D/2 STEP D
      Dx=K(n%,1,0)+K(n%,2,0)*COS(w1)*COS(w2)+K(n%,3,0)*COS(w1)*SIN(w2)+
      K(n%,4,0)*SIN(w1)
      Dy=K(n%,1,1)+K(n%,2,1)*COS(w1)*COS(w2)+K(n%,3,1)*COS(w1)*SIN(w2)+
      K(n%,4,1)*SIN(w1)
      Dz=K(n%,1,2)+K(n%,2,2)*COS(w1)*COS(w2)+K(n%,3,2)*COS(w1)*SIN(w2)+
      K(n%,4,2)*SIN(w1)
      CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
      CALL Draw&(RastPort&,Xp%,Yp%)
    NEXT w2
  NEXT w1
  FOR w1=-Pd2+D TO Pd2-D/2 STEP D
    Dx=K(n%,1,0)+K(n%,2,0)*COS(w1)+K(n%,3,0)*SIN(w1)
    Dy=K(n%,1,1)+K(n%,2,1)*COS(w1)+K(n%,3,1)*SIN(w1)
    Dz=K(n%,1,2)+K(n%,2,2)*COS(w1)+K(n%,3,2)*SIN(w1)
    CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
    CALL Move&(RastPort&,Xp%,Yp%)
```

```
    FOR w2=D TO Pm2+D/2 STEP D
      Dx=K(n%,1,0)+K(n%,2,0)*COS(w1)*COS(w2)+K(n%,4,0)*COS(w1)*SIN(w2)+
      K(n%,3,0)*SIN(w1)
      Dy=K(n%,1,1)+K(n%,2,1)*COS(w1)*COS(w2)+K(n%,4,1)*COS(w1)*SIN(w2)+
      K(n%,3,1)*SIN(w1)
      Dz=K(n%,1,2)+K(n%,2,2)*COS(w1)*COS(w2)+K(n%,4,2)*COS(w1)*SIN(w2)+
      K(n%,3,2)*SIN(w1)
      CALL Projektion(Xp%,Yp%,Dx,Dy,Dz)
      CALL Draw&(RastPort&,Xp%,Yp%)
    NEXT w2
  NEXT w1
RETURN
```

Vielleicht noch ein Wort zur Kreisdarstellung. Die Routinen die mit Kreisen zu tun haben - das sind eigentlich alle außer 'ZeichneEbene', 'ZeichneViereck' und 'ZeichneDreieck' - berechnen die Punkte der Kreisperipherie immer auf die gleiche Art und Weise: In Abhängigkeit eines Winkels werden verschiedene Stützpunkte der Peripherie berechnet. Dabei gilt: Je mehr Stützpunkte, desto runder der Kreis.

So wird klar, das feinere Rundungen gleichbedeutend mit längeren Rechenzeiten sind. Wir haben uns deshalb entschieden, die Anzahl der Stützpunkte eines Kreises offenzulassen. Später beim Schattieren erhalten Sie aber immer völlig runde Kreise, egal wie viele Stützpunkte Sie bei den Drahtmodellen verwenden.

Die Variable 'AnzahlSegmente' bestimmt vielmehr, wie rund ein Kreis wird:



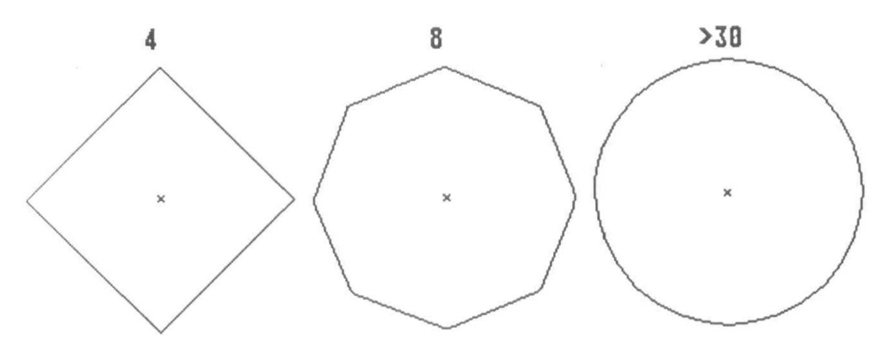


Abbildung 4.18

Doch zurück zur Körper- und Flächendarstellung:

Die oben aufgelisteten 'Zeichne..'-Routinen sind noch ziemlich isoliert. Wer ruft die einzelnen Routinen denn auf? Nun, auch daran haben wir gedacht. Folgende Routine springt in Abhängigkeit von 'K(n%,0,0)' die zugehörige 'Zeichne'-Routine an:

```
ZeichneAlle:
  ' Zeichenroutine fuer jeden einzelnen Körper aufrufen
  FOR n%=1 TO AnzahlK
    IF K(n%,0,0)=0 THEN
      GOSUB ZeichneEbene
    END IF
    IF K(n%,0,0)=1 THEN
      GOSUB ZeichneDreieck
    END IF
    IF K(n%,0,0)=2 THEN
      GOSUB ZeichneViereck
    END IF
    IF K(n%,0,0)=3 THEN
      GOSUB ZeichneKreis
    END IF
    IF K(n%,0,0)=4 THEN
      GOSUB ZeichneKreisSektor
    END IF
    IF K(n%,0,0)=5 THEN
```

```
      GOSUB ZeichneKreisRing
    END IF
    IF K(n%,0,0)=10 THEN
      GOSUB ZeichneKugel
    END IF
    IF K(n%,0,0)=20 THEN
      GOSUB ZeichneZylinder
    END IF
    IF K(n%,0,0)=21 THEN
      GOSUB ZeichneZylinderSegm
    END IF
    IF K(n%,0,0)=22 THEN
      GOSUB ZeichneKegel
    END IF
    IF K(n%,0,0)=24 THEN
      GOSUB ZeichneEllipsoid
    END IF
  NEXT n%
RETURN
```

Damit außerdem dafür gesorgt wird, daß z.B. der Bildschirm gelöscht wird, bevor alle Objekte neugezeichnet werden, haben wir die Routine 'ZeichneNeu' entwickelt, die alle nötigen Schritte vor dem Aufruf von 'ZeichneAlle' übernimmt (Anschalten des Display-Screens, Zeichnen und auf eine Taste warten und wieder zum Benutzer-Screen zurückkehren).

Wurde zwischenzeitlich keiner der Parameter 'Projektionspunkt', 'Hauptpunkt', etc. verändert, wird das Bild nicht neu gezeichnet, sondern einfach nur gezeigt.

Sollten Sie einen Hintergrund für das Bild ausgewählt haben, weil Ihnen die Hintergrundfarbe zu trist war, so sorgt 'ZeichneNeu' auch dafür, das bei ausgewähltem Hintergrund der Bildschirm nicht gelöscht wird - sonst wäre der Hintergrund ja 'futsch'! Das Bild wird dann über diesen Hintergrund gezeichnet.

```
ZeichneNeu:
  IF (Hg = False) AND (Neu! = True) THEN  ' kein Hintergrund
    CALL SetRast&(RastPort&,0)            ' aber Neuzeichnen
  END IF                                  ' => Bild löschen
```

```
    CALL Scron

    IF (Neu! = True) OR (Hg = True) THEN     ' Hintergrund und Neu-
                                             ' zeichnen
      GOSUB DeleteMenu
      RetRast& = RastPort&                   ' RastPort retten
      RastPort& = WINDOW(8)                  ' In Window-RastPort
                                             ' (GIMMEZERO Window!)
      CALL SetAPen&(RastPort&,1)             ' des neuen Screens
      GOSUB ZeichneAlle                      ' zeichnen (wegen
      Clipping)
      RastPort& = RetRast&
      BEEP
    END IF

    IF WartFlg! = True THEN                  ' Auf Taste oder Maus
                                             ' warten
       GOSUB Warte
       CALL Scroff
    END IF

    IF (Neu! = True) OR (Hg = True) THEN
      Neu! = False                           ' Bild wurde neu
      Hg = False                             ' gezeichnet, und
      GOSUB MakeMenu                         ' Hintergrund
      "uebermalt"
    END IF
  RETURN
```

4.2. Der Editor

4.2.1 Warum überhaupt ein Editor?

Unser Ziel ist es, ein Programm zu entwerfen, das uns beliebige Bilder berechnet, die sich aus einfachen Körpern zusammensetzen lassen. Welche Körper und welche Daten hierfür nötig sind, wurde schon in den voranstehenden Kapiteln beschrieben. Auch schon bekannt ist, daß diese Daten in den Arrays k! und

mat! abgespeichert werden. Die Frage ist bloß, wie stopfen wir die Daten für unsere Bilder in diese Felder? Verschiedene Methoden sind denkbar.

So wäre eine Lösung, die Daten als Data-Zeilen an das Programm anzuhängen und mittels einer entsprechenden Routine einzulesen. Dies ist sicherlich die einfachste, wenn auch für den Anwender komplizierteste. Einziger Vorteil: man kann sich eine Menge Zeit für die Programmentwicklung sparen. Nachteil: der Benutzer muß ständig das Programmm verändern, muß schon vorher das fertige Bild vor Augen und dann von Hand die benötigten Daten berechnet haben, und schließlich hat er ebenfalls keine Kontrolle darüber, ob seine Daten überhaupt seinen Vorstellungen entsprechen.

Eine weitere Möglichkeit bestünde darin, mit Hilfe eines Textverarbeitungsprogrammes Zahlenkolonnen abzuspeichern und im Programm wieder einzulesen. Die oben genannten Nachteile bleiben fast vollständig für diesen Lösungsansatz bestehen. Eine Eingabe im Dialog ist zwar etwas komfortabler, aber immer noch nicht das Gelbe vom Ei.

Besser wäre es, die Werte in einem Menü abzufragen, und diese Zahlen sofort grafisch umzusetzen. Hierdurch hat der Anwender einerseits ein sehr einfach zu bedienendes Programm vor sich, falls die Menüeingabe dementsprechend programmiert wurde, und zugleich die Möglichkeit, sein Bild schon vorher schematisch dargestellt zu sehen, um dann eventuell noch einige Korrekturen vorzunehmen. Wie dies nun im einzelnen aussehen und in BASIC programmiert werden kann, wird in den nun folgenden Kapiteln gezeigt.

4.2.2 Aufgaben und Einzelheiten des Editors

Nachdem das grobe Ziel festgelegt wurde, steht uns noch die gesamte Feinarbeit bevor, es müssen also sämtliche wünschenswerte Funktionen und Eigenschaften dieses Editors genau festgelegt werden. Hierzu zerteilen wir den gesamten Editor in seine einzelnen, logisch und funktional zummengehörenden Teile,

Module genannt. Die Hauptaufgabe des Editors ist es ja, numerische Daten einzulesen. Hierfür brauchen wir also geeignete Einlese- und Ausgabe-Prozeduren. Da diese Daten nicht nur als Zahlen dargestellt werden sollen, sondern auch als Drahtmodell, müssen auch hierzu die entsprechenden Prozeduren entwickelt werden. Schließlich werden noch einige zusätzliche Funktionen benötigt, die zwar nicht unbeding notwendig sind, aber das Programm komfortabler gestalten. Diese einzelnen Module werden nun in den folgenden Unterpunkten genauer erläutert und einige wichtige Kriterien schon herausgearbeitet. Die gesamte Feinarbeit folgt aber erst in den sich anschließenden Kapiteln.

Die grafische Ausgabe

Es ist klar, daß die grafische Darstellung nicht in Form eines vollständig mit Licht, Schatten, Hidden-Lines und Spiegelungen versehenen Bildes erfolgen kann, da die Rechenzeit viel zu groß ist. Ebenfalls möglich, aber nicht vorteilhaft ist die Ausgabe in Form eines perspektivischen Drahtmodells, da durch die Perspektive die Körper verzerrt abgebildet werden und Details nur schlecht erkennbar sind. Diese Darstellungsform ist nur zur letzten Kontrolle geeignet. Die einzige Möglichkeit, die uns für die Eingabe sinnvoll erschien, war die Projektion der Körper auf die drei Hauptebenen. Dies hört sich kompliziert an, aber ist im Grunde sehr einfach: gezeigt wird die Szenerie in Vorder-, Seiten- und Draufsicht, also wie bei einer technischen Zeichnung. Die Vorderansicht entspricht nun der Projektion auf die xz-Ebene, die Seitenansicht der Projektion auf die yz-Ebene und schließlich die Draufsicht der Projektion auf die xy-Ebene.

Beispiel:

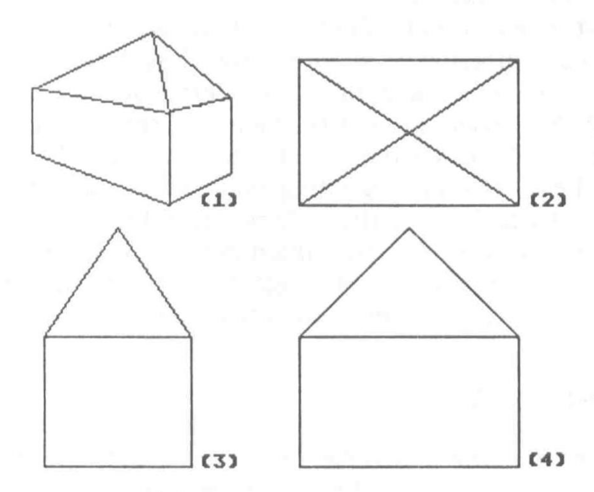


Abbildung 4.19

Diese drei verschiedenen Ansichten werden sinnvollerweise in drei Fenstern angezeigt, und zwar gleichzeitig. Da sich der Bildschirm nur sehr unvorteilhaft dritteln läßt, vierteln wir ihn lieber und haben somit sogar noch ein Fenster übrig, in dem die textuellen Ein- und Ausgaben geschehen. Da ein Fenster somit nur sehr klein sein kann, muß es vergrößerbar und verschiebbar sein. Zusätzlich muß es noch möglich sein, den Ausschnitt, der in den einzelnen Fenstern sichtbar ist, zu verschieben und zu vergrößern, damit auch größere Szenerien eingegeben und Details erkannt werden können. Auch dies soll durch ein Beispiel verdeutlicht werden:

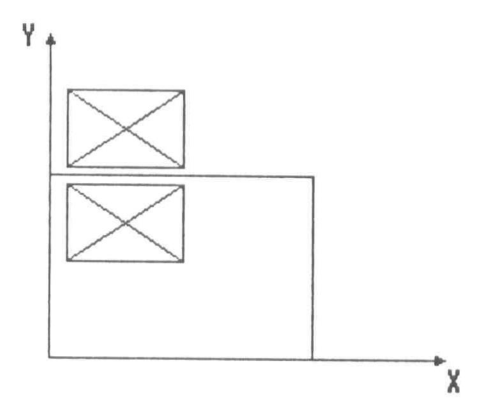


Abbildung 4.20

Wird der Ausschnitt so gewählt, bleibt das zweite Haus im verborgenen. Durch verschieben des Rahmens in y-Richtung, wird nun auch das zweite Haus sichtbar:

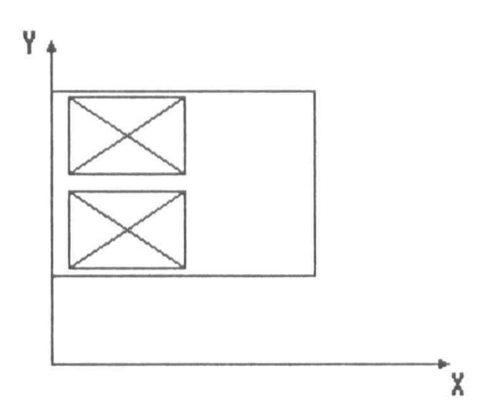


Abbildung 4.21

Der Ausschnitt wird dabei durch die Position der linken oberen Ecke eindeutig bestimmt. Im ersten Fall liegt diese Ecke bei (0,28,0) und im zweiten Fall bei (0,42,0). Hiermit wäre die Konzeption zu diesem Punkt abgeschlossen.

Die Datenein- und -ausgabe

Ein Programm sollte immer so gestaltet sein, daß auch der unerfahrene Anwender schnell damit zurechtkommt, Fehleingaben möglichst sofort erkannt werden und Korrekturen an schon eingegebenen Daten möglich sind und einfach vorgenommen werden können. Diese Eingabe erfolgt am besten in Menüs. Bezogen auf unser Programm, könnte man zum Beispiel erst folgenden Text ausgeben:

```
Körper:1  Kugel
sichtbar    :_          Material:__
Mittelpunkt:____,____,____
Radius      :____
```

und danach die einzelnen Eingabepositionen hintereinander abfragen. Hierbei muß der Programmierer möglichst dafür sorgen, daß das Menü auch von dem bösartigsten Benutzer (das ist im allgemeinen er selbst) nicht zerstört werden kann. Prinzipiell könnte diese Eingabe auch mit der INPUT-Anweisung erfolgen, doch ist dies nicht empfehlenswert, da man keine Kontrolle über die eingegebenen Zeichen hat. Besser ist es, sich seine eigene Eingaberoutine zu schreiben. Neben Korrekturen sind in unserem Programm ebenfalls Manipulationen der Daten sehr hilfreich. Unter Manipulationen verstehe ich also: Rotation, Verschieben, Vergrößerung und Kopieren eines Körpers, Funktionen, mit denen sich viel Zeit und Arbeit einsparen lassen. Natürlich muß ebenfalls vorgesehen sein, alte Datensätze wieder anzeigen zu lassen oder zu löschen. Die Anwendung des Körpereditors werden noch einfacher und komfortabler, wenn die Maus als Eingabegerät hinzugenommen wird. Mit ihr wird es dann möglich, die nötigen Koordinaten für einen Körper auf dem Bildschirm einfach "anzuklicken" und nicht wie bisher durch Ausprobieren, Ausrechnen und Eingeben in den Computer zu bekommen. Nun muß das ganze Programm noch optisch aufgepeppt und bedienerfreundlich gestaltet werden.

Zusatzfeatures

Was die meisten professionellen Progamme auf dem Amiga so auszeichnet, ist die Verwendung von Gadgets, also jenen Objekten auf dem Bildschirm, die durch die Maus bedient werden können.(z.B.: Bei dem Programm "preferences", das sich auf der Workbenchdiskette befindet, werden die Farben durch Regler eingestellt, fast so wie am Fernsehgerät, und auch sämtliche anderen Funktionen können durch einfaches "Anklicken" mit der Maus aktiviert werden. All diese Objekte heißen in der Fachsprache Gadgets.) Die Programmierung von Gadgets auf dem Amiga in BASIC ist einerseits recht umständlich, macht aber andererseits das Programm leicht bedienbar und gibt ihm auch den letzten Pfiff.

Und schließlich noch ein letzter Punkt: Was ist, wenn der Benutzer unseres Programmes vorhaben sollte, ein altes Bild, das aus 253 verschiedenen Körpern besteht, leicht abzuändern? Sollte er gezwungen sein, alle 253 Körper neu einzugeben? Wir meinen nein. Deshalb bieten wir ihm selbstverständlich noch eine Save- und Load-Funktion an.

Durch die oben aufgezählten Eigenschaften unseres Editors ist praktisch das gesamte Programm schon festgelegt, es fehlt nur noch die Realisierung. Eine Möglichkeit, einen solchen Editor in BASIC zu programmieren, wird in den folgenden Kapiteln vorgestellt, wobei jedoch nur die wichtigsten Routinen beschrieben werden. Aus diesen Bausteinen können Sie sich dann relativ leicht und schnell Ihren eigenen Editor zusammenstellen.

Als Beispiel für einen solchen Editor haben wir auf der mitgelieferten Diskette ein vollständiges Programm abgespeichert, daß alle erwähnten Funktionen besitzt. Eine kurze Bedienungsanleitung befindet sich zusätzlich im Kapitel 5.1 dieses Buches. Nach so viel Theorie, empfehle ich nun jedem Leser, sich dieses Programm ein Mal anzuschauen, um einen ungefähren Eindruck zu bekommen, was Sie auf den nächsten Seiten alles erwartet. Bevor Sie jetzt also das Buch zur Seite legen, um Ihren Amiga anzuwerfen und mit unserer Diskette zu füttern, sollte ich Ihnen vorher noch folgenden Tip geben, falls Sie Schwierigkeiten mit

den oben beschriebenen Projektionen haben. Es befindet sich nämlich auf dieser Diskette der Datensatz zum Haus-Beispiel. Um sich dieses Haus anschauen zu können, müssen Sie folgende Befehle durchführen:

1. Das AmigaBASIC laden
2. CLEAR,120000 eingeben
3. LOAD "editor" eingeben
4. Das Programm starten (mit RUN)
5. Nun nicht verzweifeln, sondern noch einige Sekunden warten
6. Im Menü "Diskette" "loadlist" auswählen
7. "haus" eintippen und RETURN drücken
8. CTRL-R drücken
9. Nun noch länger warten
10. Herumprobieren (z.B. mit Ausschnitt)

Alles klar? Dann kann es ja weiter gehen!

4.2.3 Die einzelnen Komponenten des Editors

In den nun folgenden Kapiteln werden die oben schon vorgestellten Module noch genauer behandelt, weiter verfeinert und schließlich als BASIC-Programm vorgestellt. Zunächst werden in jedem Kapitel die groben Ziele und die damit verbundenen Probleme dargestellt und die einfacheren Prozeduren erarbeitet, auf die sich dann die schrittweise komplizierter werdenden Prozeduren stützen. Im Eingabe-Modul zum Beispiel wird zunächst eine Prozedur zum Einlesen einer Taste entwickelt. Diese wird von der Einleseroutine für einen String verwendet, worauf sich wiederum die Eingabeprozedur für eine Realzahl abstützt, die schließlich noch von der Prozedur zum Einlesen eines Vektors aufgerufen wird. Dieses Prinzip des schrittweise Entwickelns immer komplexerer Algorithmen wird auch in den restlichen Kapiteln beibehalten.

4.2.3.1 Die Eingabe

Wenn in einem Programm eine Zahl eingegeben werden soll, so ist eine beliebte Eingabefehlerquelle, statt zum Beispiel "123" "einhundert23", "1a23" oder ähnliches einzutippen. Deshalb sollten die Eingaberoutinen am besten so gestaltet sein, daß nur die Tasten aktiv sind, die der Benutzer auch drücken darf. Das hieße bei unserem Zahlenbeipiel, daß wirklich nur die Tasten +,-,0,..,9 angenommen werden. Neben den erlaubten Tasten sollten natürlich "Delete","Backspace", "Cursor-Left", "Cursor-Right" und "Return" wie gewohnt funktionieren. Daß hierzu die INPUT-Routine des BASIC nicht verwendbar ist, sollte nun einsichtig sein. Zwar wird so der Programmieraufwand erhöht, doch macht sich diese Arbeit auf jeden Fall im fertigen Programm positiv bemerkbar. Die Prozedur, die anstelle vom INPUT-Befehl verwendet wird, heißt "getstring". Sie ist so konzipiert, daß ein String an einer bestimmten Stelle des Bildschirms in einem Fenster variabler Länge eingelesen wird. Innerhalb dieses Fensters kann die Zeichenkette beliebig nach links und rechts "gescrollt" (verschoben) werden.

Doch um überhaupt eine Zeichenkette einlesen zu können, muß eine Routine zur Verfügung stehen, die den Cursor auf den Bildschirm setzt, auf eine gedrückte Taste wartet und danach den Cursor wieder löscht. Um den Cursor an einer bestimmten Position des Bildschirms hinzuzeichnen, müssen vorher die Zeichenhöhe und -breite bestimmt werden, da diese vom gewählten Zeichensatz abhängen. Die hierfür benötigten Informationen stehen in der RASTPORT-Datenstruktur. Zu erklären, was es mit dieser RASTPORT-Datenstruktur auf sich hat und wie dies aufgebaut ist, würde den Rahmen dieses Buches sprengen. Wichtig für unsere Zwecke ist nur, daß die Zeichenhöhe und -breite wie folgt bestimmt werden können:

```
rp&=WINDOW(8)      'Hole den Zeiger auf den RASTPORT
cw%=PEEKW(rp&+60) 'Zeichenbreite
ch%=PEEKW(rp&+58) 'Zeichenhöhe
```

Um nun den Cursor zu zeichnen, so daß das Zeichen unter dem Cursor weiterhin lesbar bleibt, muß der entsprechende Bildschirmbereich invertiert werden. Dies geschieht am einfachsten

schirmbereich invertiert werden. Dies geschieht am einfachsten mit SETDRMD, einer Funktion, die sich in der GRAPHICS-LIBRARY befindet und die mit

```
LIBRARY "basicdemos/graphics.library"
```

in unser Programm eingebunden wird. Nach diesen Vorbereitungen ist nun die Programmierung der besprochenen Routine recht einfach:

```
SUB setcursor(z,s) STATIC
'AUFGABE  :setzt den Cursor auf die angegebene Position
'PARAMETER:=>z Zeile
'            s Spalte
  SHARED cw%,ch%
  CALL setdrmd&(WINDOW(8),2) 'Ivers-modus einstellen
  LINE (s*cw%-cw%,z*ch%-ch%-1)-(s*cw%,z*ch%-1),3,bf 'Cursor setzen
  CALL setdrmd&(WINDOW(8),1) 'Normal-modus einstellen
END SUB
```

Wird setcursor nun mit den gleichen Parametern zweimal aufgerufen, so wird der Cursor wieder gelöscht, und das Zeichen darunter ist wieder normal dargestellt:

```
SUB getkey(a$,z,s) STATIC
'AUFGABE  :Liest einen Charakter ein
'PARAMETER:=>z Zeile
'            s Spalte
'          <=a$ gedrueckte Taste
  CALL setcursor(z,s) 'den Cursor setzen
  a$=""
  WHILE a$=""  'auf Taste warten
    a$=INKEY$
  WEND
  CALL setcursor(z,s)  'den Cursor löschen
END SUB
```

Auf diese beiden Prozeduren stützt sich nun die Prozedur getstring, die als Parameter den alten Inhalt des Strings, die schon gedrückte Taste, die erlaubten Zeichen, Fensterpositon und Fensterlänge verlangt. Durch die Angabe des alten String-Inhalts kann eine schon eingegebene Zeichenkette partiell verändert werden, ohne daß der gesamte Inhalt neu eingegeben

länge von Eins nur auf eine gültige Taste gewartet wird und nicht noch auf ein abschließendes Return.

```
SUB getstring(old$,a$,z$,z,s,laenge) STATIC
'AUFGABE  :Einlesens eines Strings,wobei die Eingabezeile
'          voll editierfähig ist d.h.:Cursor left/right,
'          Backspace und Delete funktionieren.
'          Außerdem wird der String gescrollt, falls die
'          Eingabe länger als vorgesehen ist.
'PARAMETER:=>old$ alter Wert des einzulesenen Strings
'            a$ die gedrueckte Taste
'            z$ die erlaubten Zeichen
'            z Zeile
'            s Spalte
'            laenge Länge des Eingabefensters
'          <=old$ der eingegebene String
  IF laenge=1 THEN 'soll nur 1 Zeichen eingegeben werden?
    old$=a$        '                        =>kein Return
    WHILE INSTR(z$,old$)=0  'Warten auf gültiges Zeichen
      CALL getkey(old$,z,s)
    WEND
    LOCATE z,s
    PRINT old$
  ELSE
    i=1         'zeigt auf die aktuelle String-Position
    position=1 'zeigt auf die aktuelle Bildschirmposition
    WHILE ASC(a$)><13  'solang, bis Return
      IF INSTR(z$,a$)><0 THEN 'erlaubtes Zeichen?
        old$=LEFT$(old$,i-1)+a$+RIGHT$(old$,LEN(old$)-i+1)
        i=i+1                'Zeichen einfügen
        position=position+1
        IF position>laenge THEN
          position=laenge
        END IF
      ELSE
        IF ASC(a$)=30 AND i<=LEN(old$) THEN  'Cursor right?
          i=i+1
          position=position+1
          IF position>laenge THEN
            position=laenge
          END IF
        ELSE
```

```
          IF ASC(a$)=31 AND i>1 THEN 'Cursor left?
            i=i-1
            position=position-1
            IF position=0 THEN
              position=1
            END IF
          ELSE
            IF ASC(a$)=127 AND i<=LEN(old$) THEN  'delete ?
              old$=LEFT$(old$,i-1)+RIGHT$(old$,LEN(old$)-i)
            ELSE              'Zeichen löschen
              IF ASC(a$)=8 AND i>1 THEN  'Backspace ?
                i=i-1
                position=position-1
                IF position=0 THEN
                  position=1
                END IF
                old$=LEFT$(old$,i-1)+RIGHT$(old$,LEN(old$)-i)
              END IF            'Zeichen löschen
            END IF
          END IF
        END IF
      END IF
      LOCATE z,s  'String-Ausgabe
      PRINT MID$(old$+" ",i-position+1,laenge)
      CALL getkey(a$,z,s+position-1) 'nächste Taste holen
    WEND
  END IF
  IF i><position THEN  'falls gescrollt wurde,Anfangsstring
    CALL putstring(old$,z,s,laenge)                'ausgeben
  END IF
END SUB
```

Zunächst wird untersucht, ob laenge gleich eins ist oder nicht. Im ersten Fall wird nur auf eine gültige Taste gewartet und diese zurückgegeben, im anderen Fall werden solange Zeichen eingelesen, bis Return gedrückt wird. Bei jedem Zeichen wird geprüft, ob einer der folgenden fünf Fälle vorliegt:

1. Gültiges Zeichen (d.h.:es kommt in z$ vor), dann ist natürlich INSTR(z$,a$) ungleich Null und das Zeichen wird an der entsprechenden Stelle in old$ eingefügt. Anschließend müssen nur noch i und position um eins erhöht werden.

2. Cursor-Right, dann werden nur i und position um eins erhöht.
3. Cursor-Left, hier werden i und position um eins erniedrigt, wobei jedoch, wie in den vorherigen Fällen, darauf zu achten ist, daß diese beiden Variablen keine ungültigen Werte annehmen.
4. Delete, hier wird im String old$ nur das i-te Zeichen herausgeschnitten.
5. Backspace, dies ist praktisch die Kombination aus Cursor-Left und Delete.

Anschließend wird der gesamte gültige Bereich des Strings ausgegeben und auf die nächste Taste gewartet. Wurde schließlich Return gedrückt, so werden die ersten laenge Zeichen ausgegeben, falls diese nicht sichtbar sein sollten. Hierzu wird die Routine putstring verwendet, die erst später beschrieben wird.

Mit Hilfe dieser Routine werden natürlich die Einleseoperationen für Integer- und Realwerte sehr vereinfacht:

```
SUB getreal(i!,a$,z,s,laenge,unt!,ob!) STATIC
'AUFGABE  :Liest einen Realwert ein
'PARAMETER:=>i! alter Wert
'             a$ gedrueckte Taste
'             z Zeile
'             s Spalte
'             laenge Länge der Eingabezeile
'             unt! untere Schranke
'             ob! obere Schranke
'           <=i! eingegebener Wert
  CALL conrealstr(i!,j$)  'Konvertierung von Real nach String
loop1:
  i$=j$
  CALL getstring(i$,a$,"1234567890-.",z,s,laenge)
  j$=i$
  CALL constrreal(i$,i!)  'Umwandlung nach Real
  IF i$="" OR i!<unt! OR i!>ob! THEN  'solange versuchen bis
    a$=" "                            'gültige Eingabe
    GOTO loop1
  END IF
END SUB
```

Die Routine zum Einlesen eines Integer-Wertes ist fast identisch zu der oben abgedruckten Prozedur, sie unterscheidet sich bloß im getstring-Aufruf (bei getreal ist "." erlaubt, bei getint nicht). Zu den schon bekannten Parametern ist hier noch eine Angabe des zulässigen Wertebereichs nötig, also die untere und die obere Grenze aller gültigen Zahlen (unt!,ob!). Beide Prozeduren verwenden auch zwei Routinen, die bisher noch nicht beschrieben wurden: die Konvertierungsroutinen conrealstr und constrreal:

```
SUB conrealstr(i!,i$)STATIC
'AUFGABE  :konvertiert einen Realwert in einen String
'          der Wert wird dabei auf 3 Nachkommastellen gerundet.
'PARAMETER:=>i! der Realwert
'          <=i$ der String
  i$=STR$(i!)
  j!=FIX(1000!*i!+.5*SGN(i!))/1000! 'runden
  i$=STR$(j!)
  IF ABS(j!)>0 AND ABS(j!)<1 THEN  'falls 0<j!<1 0 anfuegen
    i$=MID$(i$,1,1)+"0"+MID$(i$,2,LEN(i$))
  END IF
  IF j!>=0 THEN  'falls 0<=j! erste Stelle abschneiden
    i$=RIGHT$(i$,LEN(i$)-1)
  END IF
END SUB

SUB constrreal(i$,i!) STATIC
'AUFGABE  :konvertiert einen String in eine Realzahl
'PARAMETER:=>i$ der String
'          <=i! der Realwert
'            i$ falls Konvertierungsfehler i$=""
  i!=VAL(i$)
  IF i!>=0 THEN
    IF i!>0 AND i!<1 THEN
      i$=" "+RIGHT$(i$,LEN(i$)-1) 'ein Leerfeld anfuegen und
    ELSE                          'die Null abschneiden
      i$=" "+i$  'ein Leerfeld anfuegen
    END IF
  END IF
```

```
    j!=i!-FIX(i!*1000!)/1000! 'runden
    IF i$><STR$(i!) OR j!><0 THEN 'Fehler ?
      i$=""
    END IF
  END SUB
```

Bei beiden Prozeduren ist darauf zu achten, daß Realwerte in diesem Programm nur drei Nachkommastellen haben dürfen, und daß Werte, deren Betrag zwischen Null und Eins liegt, eine vorangestellte Null haben müssen. Erfüllt eine Eingabe nicht diese Bedingungen, so liefert die constrreal-Routine für **i$** die leere Zeichenkette als Ausgabewert, als Zeichen einer falschen Eingabe. Die Einschränkung auf drei Nachkommastellen ist zwar nicht unbedingt nötig, doch genauere Eingabewerte verändern das später zu berechnende Bild ohnehin nicht bemerkbar, so daß eine höhere Genauigkeit ohnehin sinnlos ist.

Neben Zeichenketten, Integer- und Realzahlen werden in unserem Programm besonders häufig Vektoren als Eingabe verlangt. Deshalb ist es praktisch, für diese Objekte eigene Eingabeprozeduren zu schreiben. Ein Vektor wird dabei dargestellt durch drei Real-Zahlen, jeweils voneinander getrennt durch ein Komma. Bei der Eingabeprozedur get3real verwenden wir wie gehabt wieder entsprechende Konvertierungsroutinen (con3realstr und constr3real), und zum ersten Mal taucht hier die Mausposition auf, da ja Mauseingaben erlaubt sein sollen. Sie ist natürlich nur dann sinnvoll, wenn Raumkoordinatenvektoren einzulesen sind und nicht zum Beispiel Winkelvektoren. Ob nun die Mauseingabe erlaubt ist oder nicht, wird der Prozedur durch einen zusätzlichen Parameter mitgeteilt. Die Mausposition wird am besten in globalen Variablen abgespeichert, sie heißen hier mox!, moy!, moz! (wie diese von der Maus eingelesen werden, ist eine andere Geschichte und soll ein andermal erzählt werden). Damit die Prozedur weiß, daß jetzt nicht die Tastatureingabe zählt, sondern die Mauseingabe, vereinbaren wir nun, daß dies genau dann der Fall ist, wenn nur Return gedrückt wurde und sonst keine andere Taste. Und nun die besprochene Routine:

```
SUB get3real(i1!,i2!,i3!,a$,z,s,laenge,unt!,ob!,modus) STATIC
'AUFGABE  :Liest 3 Realwerte ein
'PARAMETER:=>i1!,i2!,i3! alte Werte
'            a$ gedrueckte Taste
'            z Zeile
'            s Spalte
'            laenge Länge der Eingabezeile
'            unt! untere Schranke
'            ob! obere Schranke
'            modus falls=0  dann keine Mauseingabe
'              falls=-1 dann Mauseingabe möglich (Ortsvektor)
'              falls>0  dann Mauseingabe möglich (Differenzvektor)
'          <=i1!,i2!,i3! eingegebene Werte
  SHARED mox!,moy!,moz!,k!()
  CALL con3realstr(i1!,i2!,i3!,j$)
  IF a$="" THEN
    CALL getkey(a$,z,s)
  END IF
  IF ASC(a$)=13 AND modus><0 THEN
    IF modus=-1 THEN
      i1!=mox!
      i2!=moy!
      i3!=moz!
    ELSE
      i1!=mox!-k!(modus,1,0)
      i2!=moy!-k!(modus,1,1)
      i3!=moz!-k!(modus,1,2)
    END IF
    CALL put3real(i1!,i2!,i3!,z,s,26)
  ELSE
loop2:
    i$=j$
    CALL getstring(i$,a$,"1234567890-.,",z,s,laenge)
    j$=i$
    CALL constr3real(i$,i1!,i2!,i3!)
    IF i$="" OR i1!<unt! OR i2!<unt! OR i3!<unt! OR i1!>ob! OR i2!>ob!
    OR i3!>ob! THEN
      a$=" "
      GOTO loop2
    END IF
  END IF
END SUB
```

Das einzig Neue an dieser Routine ist die zweite IF-Abfrage, wo auch zum ersten Mal das Array k! auftaucht. Hier wird anfangs abgefragt, ob eine Mauseingabe überhaupt zulässig ist (modus ist dann ungleich Null), und wenn ja, ob der Mausvektor oder der Differenzenvektor zwischen dem Mausvektor und dem Bezugsvektor des Körpers mit dem Index modus übernommen werden soll. Wie auch vorher schon, verwendet diese Routine wieder eigene Konvertierungsprozeduren (con3realstr und constr3real, auf die Prozedur put3real wird wieder später eingegangen):

```
SUB con3realstr(i1!,i2!,i3!,i$) STATIC
'AUFGABE  :Konvertiert 3 Realwerte in einen Vektorstring
'PARAMETER:=>i1!,i2!,i3! die 3 Realwerte
'          <=i$ der String
  CALL conrealstr(i1!,i1$)
  CALL conrealstr(i2!,i2$)
  CALL conrealstr(i3!,i3$)
  i$=i1$+","+i2$+","+i3$
END SUB

SUB constr3real(i$,i1!,i2!,i3!) STATIC
'AUFGABE  :Konvertiert einen String in 3 Realwerte
'PARAMETER:=>i$ der String
'          <=i1!,i2!,i3! die Realwerte
'            i$ falls Konvertierungsfehler i$=""
  fehler=0
  i$=i$+","
  FOR i=1 TO 3
    komma=INSTR(i$,",") 'Kommaposition merken
    IF komma<=1 THEN    'Fehler ?
      fehler=1
    ELSE  'den zu konvertierenden String herausschneiden
      a$=MID$(i$,1,komma-1)
      i$=RIGHT$(i$,LEN(i$)-komma) 'den Rest merken
      CALL constrreal(a$,i!(i))   'konvertieren
      IF a$="" THEN  'Fehler ?
        fehler=1
      END IF
    END IF
  NEXT i
  IF fehler=1 THEN
    i$=""
  ELSE
```

```
      i$=" "
      i1!=i!(1)
      i2!=i!(2)
      i3!=i!(3)
    END IF
  END SUB
```

Beide Routinen stützen sich auf die schon vorher entwickelten Konvertierungsprozeduren und funktionieren im Prinzip genauso. Nur die Prozedur constr3real ist ein wenig komplizierter, da jeweils die Anfangsposition der einzelnen Koordinaten in der Zeichenkette festgestellt werden muß, was aber mittels der INSTR-Funktion recht einfach ist. Nach so viel Vorarbeit wollen wir uns nun daran begeben, ein Menü für unseren Editor zu programmieren. Als Beispiel wird hier das Menü für den Körper Kreisteil besprochen, da dort alle oben entwickelten Eingabeprozeduren verwendet werden. Zunächst muß man sich darüber im klaren sein, wo auf dem Bildschirm was und mit welcher Länge stehen darf. Diese Positionen sind recht willkürlich und dem Geschmack des einzelnen entsprechend verschieden. Um das folgende Programm einfach zu halten, werden zweckmäßigerweise diese Werte in einem Array (tabs) abgelegt.

Eine Forderung an unsere Menüs war, daß zu jeder Zeit eine Korrektur auch der schon vorher eingegebenen Zeilen möglich ist. Dies soll nun mit den Tasten Cursor-Up und Cursor-Down geschehen. Mit Cursor-Up gelangt man zur vorherigen Eingabeposition, mit Cursor-Down zur nächsten. Damit die Prozedur weiß, welche Eingabe als nächste erfolgt, wird eine entsprechende Zählvariable (nr) verwendet, die nur Werte in einem bestimmten Bereich annehmen darf (0 <= nr <= Anzahl der Eingabepositionen). Neben den reinen Körperdaten wird noch eine zusätzliche Information benötigt, und zwar, ob der Körper nachher in den Projektionsfenstern zu sehen sein soll oder nicht. Dieser Boolean-Wert wird in k!(ptr,5,2) abgespeichert, wobei ptr auf den einzugebenen Körper zeigt und der Typ schon in k!(ptr,0,0) eingetragen wurde.

```
SUB getkrteil(ptr) STATIC
  SHARED k!(),tabs(),min!,max!,grad!,rad!
  tabs(0,0)=2     'grad! und rad! dienen nur der Umwandlung von
  tabs(1,0)=2     'einem Winkelsystem in das andere
  tabs(2,0)=3     'min! und max! sind globale Konstanten und
  tabs(3,0)=4     'legen den allgemeinen Wertebereich fest
  tabs(4,0)=5
  tabs(5,0)=6
  tabs(6,0)=6
  tabs(7,0)=7
  tabs(8,0)=7
  tabs(0,1)=10
  tabs(1,1)=22
  tabs(2,1)=4
  tabs(3,1)=4
  tabs(4,1)=4
  tabs(5,1)=4
  tabs(6,1)=16
  tabs(7,1)=4
  tabs(8,1)=16
  nr=0
  WHILE nr<=8
    CALL getkey(a$,tabs(nr,0),tabs(nr,1))
    IF ASC(a$)=28 THEN
      IF nr>0 THEN
         nr=nr-1
      END IF
    ELSE
      IF ASC(a$)=29 THEN
        nr=nr+1
      ELSE
        IF nr=0 THEN
          CALL getstring(b$,a$,"jn",tabs(nr,0),tabs(nr,1),1)
          IF b$="j" THEN
            k!(ptr,5,2)=1
          ELSE
            k!(ptr,5,2)=0
          END IF
        END IF
        IF nr=1 THEN
          CALL getint(k!(ptr,0,2),a$,tabs(nr,0),tabs(nr,1),5,
          1!,max!)
        END IF
        IF nr=2 THEN
          CALL get3real(k!(ptr,1,0),k!(ptr,1,1),k!(ptr,1,2),a$,
          tabs(nr,0),tabs(nr,1),26,min!,max!,-1)
```

```
        END IF
        IF nr=3 THEN
          CALL get3real(k!(ptr,2,0),k!(ptr,2,1),k!(ptr,2,2),a$,
          tabs(nr,0),tabs(nr,1),26,min!,max!,ptr)
        END IF
        IF nr=4 THEN
          CALL get3real(k!(ptr,3,0),k!(ptr,3,1),k!(ptr,3,2),a$,
          tabs(nr,0),tabs(nr,1),26,min0!,max!,ptr)
        END IF
        IF nr=5 THEN
          k!=grad!*k!(ptr,5,0)
          CALL getreal(k!,a$,tabs(nr,0),tabs(nr,1),8,min!,max!)
          k!(ptr,5,0)=rad!*k!
        END IF
        IF nr=6 THEN
          k!=grad!*k!(ptr,5,1)
          CALL getreal(k!,a$,tabs(nr,0),tabs(nr,1),8,min!,max!)
          k!(ptr,5,1)=rad!*k!
        END IF
        IF nr=7 THEN
          CALL getreal(k!(ptr,4,0),a$,tabs(nr,0),tabs(nr,1),8,
          0!,1!)
        END IF
        IF nr=8 THEN
          CALL getreal(k!(ptr,4,1),a$,tabs(nr,0),tabs(nr,1),8,
          0!,1!)
        END IF
        nr=nr+1
      END IF
    END IF
  WEND
END SUB
```

Wie Sie sehen, ist die Implementierung eines Menüs in BASIC mit diesen Prozeduren recht einfach geworden. Die Einleseprozeduren für die restlichen Körper werden analog programmiert, was eigentlich keine Schwierigkeit sein dürfte. Wer dabei dennoch Probleme hat, sollte sich den fertigen Editor auf der Diskette anschauen. Damit wären nun sämtliche benötigten Eingabeprozeduren beschrieben, also auf zum nächsten, einfacheren Kapitel.

4.2.3.2 Die Ausgabe

Sinn und Zweck der Ausgaberoutinen ist es, an einer bestimmten Stelle auf dem Bildschirm den Wert einer Variablen mit einer bestimmten Länge auszugeben. Der einfachste Fall ist hier die Ausgabe einer Zeichenkette, bei der mit Hilfe der MID$-Funktion der interessierende Teil herausgeschnitten wird. Zu beachten ist nur, daß vorher der angegebene Bereich auf dem Bildschirm gelöscht wird:

```
SUB putstring(s$,z,s,laenge) STATIC
'AUFGABE  :Gibt einen String mit einer vorgegebenen Länge aus
'PARAMETER:=>s$ der String
'            z Zeile
'            s Spalte
'            laenge Länge des Ausgabefensters
  LOCATE z,s              'den angegebenen Bereich löschen
  PRINT SPACE$(laenge)
  LOCATE z,s              'den String ausgeben
  PRINT MID$(s$,1,laenge)
END SUB
```

Wie bei den Eingabeprozeduren können nun mit dieser putstring-Routine die fehlenden Operationen für Realwerte und Vektoren stark vereinfacht werden. Zuerst werden die Werte mit den Konvertierungsfunktionen in einen String verwandelt, dann mit der putstring-Routine ausgegeben:

```
SUB putreal(i!,z,s,laenge) STATIC
'AUFGABE  :Gibt einen Realwert mit einer vorgegebenen Länge aus
'PARAMETER:=>i! der Realwert
'            z Zeile
'            s Spalte
'            laenge Länge des Ausgabefensters
  CALL conrealstr(i!,s$)
  CALL putstring(s$,z,s,laenge)
END SUB
```

```
SUB put3real(i1!,i2!,i3!,z,s,laenge) STATIC
'AUFGABE  :Gibt 3 Realwerte mit einer vorgegebenen Länge aus
'PARAMETER:=>i1!,i2!,i3! die Realwerte
'             z Zeile
'             s Spalte
'             laenge Länge des Ausgabefensters
  CALL con3realstr(i1!,i2!,i3!,i$)
  CALL putstring(i$,z,s,laenge)
END SUB
```

Jetzt fehlt nur noch eine Ausgabeprozedur für die Daten eines Körpers. Im Unterschied zur Eingabe, bei der für jeden Körper eine separate Routine hergestellt werden mußte, kann bei der Ausgabe ruhig alles in einer Prozedur geschehen:

```
SUB showelem(ptr) STATIC
'AUFGABE  :Gibt die Daten des ptr'ten Elementes aus
'PARAMETER:=>ptr Zeiger auf das auszugebende Element
  SHARED k!(),typ$(),leer$,grad!,rad!
  LOCATE 1,1
  FOR i=1 TO 7  'einen Teil des Fensters löschen
    PRINT leer$
  NEXT i
  LOCATE 1,1
  PRINT "Körper:"
  CALL putreal(ptr*1!,1,8,5)
  typ=INT(k!(ptr,0,0))
  IF ptr><0 THEN
    LOCATE 1,14
    PRINT typ$(typ)
    LOCATE 2,1
    PRINT "sichtbar:";
    IF k!(ptr,5,2)=0 THEN
      PRINT "n"
    ELSE
      PRINT "j"
    END IF
    LOCATE 2,13
    PRINT "Material:"
    CALL putreal(k!(ptr,0,2),2,22,5)
    IF typ=10 THEN
      LOCATE 3,1
      PRINT "M :"
```

```
      PRINT "r :"
      CALL put3real(k!(ptr,1,0),k!(ptr,1,1),k!(ptr,1,2),3,4,26)
      CALL putreal(k!(ptr,2,0),4,4,8)
    ELSE
      LOCATE 3,1
      PRINT "M :"
      PRINT "A :"
      PRINT "B :"
      CALL put3real(k!(ptr,1,0),k!(ptr,1,1),k!(ptr,1,2),3,4,26)
      CALL put3real(k!(ptr,2,0),k!(ptr,2,1),k!(ptr,2,2),4,4,26)
      CALL put3real(k!(ptr,3,0),k!(ptr,3,1),k!(ptr,3,2),5,4,26)
      IF typ>=4 THEN
        IF typ>=20 THEN
          LOCATE 6,1
          PRINT "C :"
          CALL put3real(k!(ptr,4,0),k!(ptr,4,1),k!(ptr,4,2),6,4,26)
          IF typ=21 THEN
            LOCATE 7,1
            PRINT "sw:"
            LOCATE 7,13
            PRINT "ew:"
            CALL putreal(grad!*k!(ptr,5,0),7,4,8)
            CALL putreal(grad!*k!(ptr,5,1),7,16,8)
          END IF
        ELSE
          LOCATE 6,1
          PRINT "sw:"
          LOCATE 6,13
          PRINT "ew:"
          CALL putreal(grad!*k!(ptr,5,0),6,4,8)
          CALL putreal(grad!*k!(ptr,5,1),6,16,8)
          IF typ=5 THEN
            LOCATE 7,1
            PRINT "ri:"
            LOCATE 7,13
            PRINT "ra:"
            CALL putreal(k!(ptr,4,0),7,4,8)
            CALL putreal(k!(ptr,4,1),7,16,8)
          END IF
        END IF
      END IF
    END IF
  END IF
END SUB
```

Diese Prozedur ist eigentlich recht einfach, erwähnt werden müßte eigentlich nur noch, daß die CLS-Operation zum Löschen des Fensters nicht verwendet wurde, da ein Teil des Bildschirms weiterhin bestehen bleiben muß (bei meinem Beispieleditor sind dies die Mausposition, der Vergrößerungsfaktor und einige anderen Werte). Um diesen Teil zu löschen, wird jede Zeile mit einer bestimmten Anzahl von Leerfeldern überdruckt, diese Leerfelder stehen in leer$. Ein weiterer kritischer Punkt ist das Array typ$, in dem die Körpernamen zu den einzelnen Typnummern stehen. Das Array typ$ wurde dabei so initialisiert:

```
typ$(0)="Ebene"
typ$(1)="Dreieck"
typ$(2)="Parallelogramm"
typ$(3)="Kreis"
typ$(4)="Kreissegment"
typ$(5)="Kreisteil"
typ$(10)="Kugel"
typ$(20)="Zylinder"
typ$(21)="Zylindersegment"
typ$(22)="Kegel"
typ$(24)="Sphäroid"
```

Damit wäre auch die letzte Ausgabeprozedur besprochen.

4.2.3.3 Die Grafik

Bei den grafischen Befehlen in diesem Programm ist zu berücksichtigen, daß jeder Körper in drei verschiedenen Fenstern ausgegeben werden muß. Und nicht nur das erschwert die Implementierung dieser Routinen, man muß ebenfalls den Vergrößerungsfaktor und den gewählten Ausschnitt mit einkalkulieren. Selbst das reicht noch nicht, da die Punktbreite auf dem Amiga geringer ist als die Punkthöhe. Dies kann durch Einführung eines Korrekturfaktors für die x-Koordinaten ausgeglichen werden. Vielleicht haben Sie dieses Phänomen schon selbst entdeckt, wenn nicht, so können Sie ein Mal versuchen, ein Quadrat mit der Kantenlänge 100 auf den Bildschirm zu zeichnen. Falls Sie den Befehl:

```
LINE (0,0)-(100,100),,b
```

verwendet haben, dürfte nun ein Rechteck auf Ihrem Bildschirm zu sehen sein. Um den Korrekturfaktor zu berechnen, müssen Sie die Höhe des Rechtecks messen und durch die Länge teilen. Bei meinem Monitor kam dabei ein Wert von ungefähr 1.88 heraus, sollte er bei Ihnen anders sein, so muß er im Programm entsprechend abgeändert werden (die Variable heißt xkorrfak! und steht im Unterprogramm init).

Die einfachsten grafischen Objekte sind ein Punkt und eine Linie. In unserem Programm verwenden wir jedoch nicht Bildschirmkoordinaten, sondern Raumkoordinaten, so daß wir uns eigene Prozeduren für diese Grafikobjekte erstellen müssen. Neben der Angabe, wohin der Punkt gesetzt oder die Linie gezeichnet werden muß, benötigen Sie noch die Zeichenfarbe (farbe), die Angabe des Ausschnitts (mx!,my!,mz!) und den Vergrößerungsfaktor (faktor!). Die Information, in welches Fenster zu zeichnen ist, entnehmen die Prozeduren der Funktion WINDOW.

```
SUB setpoint(x!,y!) STATIC
'AUFGABE  :setzt einen Punkt in das aktuelle Ausgabefenster
'PARAMETER:=>x!,y! Punktkoordinaten
  SHARED mx!,my!,mz!,faktor!,xkorrfak!,farbe
  fenster=WINDOW(1)
  IF fenster=1 THEN
    LINE ((x!-mx!)*xkorrfak!*faktor!,(my!-y!)*faktor!)-
        ((x!-mx!)*xkorrfak!*faktor!,(my!-y!)*faktor!),farbe
  ELSE
    IF fenster=2 THEN
      LINE ((x!-mx!)*xkorrfak!*faktor!,(mz!-y!)*faktor!)-
          ((x!-mx!)*xkorrfak!*faktor!,(mz!-y!)*faktor!),farbe
    ELSE
      IF fenster=3 THEN
        LINE ((my!-x!)*xkorrfak!*faktor!,(mz!-y!)*faktor!)-
            ((my!-x!)*xkorrfak!*faktor!,(mz!-y!)*faktor!),farbe
      END IF
    END IF
  END IF
END SUB
```

```
SUB drawline(x!,y!)STATIC
'AUFGABE  :zeichnet vom zuletzt verwendeten Punkt bis zu dem
'          angegebenen eine Linie
'PARAMETER:=>x!,y! Punktkoordinaten
  SHARED mx!,my!,mz!,faktor!,xkorrfak!,farbe
  fenster=WINDOW(1)
  IF fenster=1 THEN
    LINE -((x!-mx!)*xkorrfak!*faktor!,(my!-y!)*faktor!),farbe
  ELSE
    IF fenster=2 THEN
      LINE -((x!-mx!)*xkorrfak!*faktor!,(mz!-y!)*faktor!),farbe
    ELSE
      IF fenster=3 THEN
        LINE -((my!-x!)*xkorrfak!*faktor!,(mz!-y!)*faktor!),
            farbe
      END IF
    END IF
  END IF
END SUB
```

Wie gewohnt kommt nun eine Prozedur, die sich auf diese beiden stützt, nämlich die Zeichenroutine für eine Ellipse. Zunächst müssen wir uns natürlich Gedanken darüber machen, wie sich überhaupt eine Ellipse konstruieren läßt.

Ich habe hier ein Verfahren verwendet, bei dem der Ellipsenmittelpunkt und zwei weitere Vektoren, die die Ellipse aufspannen, gegeben sein müssen, um eine Ellipse eindeutig festzulegen. Die Ellipse entsteht nun durch Rotation der beiden Vektoren um den Mittelpunkt und deren Addition. Wird eine Volldrehung durchgeführt, so liegen zum Schluß alle Punkte der Ellipse fest.

Da unendlich viele verschiedene Zahlen zwischen 0 und 360 liegen, ist es unmöglich, für jeden Winkel den zugehörigen Ellipsenpunkt zu berechnen. Deshalb wird jeder Kreis bzw. jede Ellipse durch eine bestimmte Anzahl (maxlinien) von Linien angenähert. Je größer die Anzahl gewählt wird, desto genauer wird die Ellipse dargestellt, aber desto länger braucht unser Programm, um eine Ellipse zu zeichnen. Darum sollte im späteren Programm diese Anzahl wählbar sein, denn so hat der Benutzer direkten Einfluß auf die Zeichengeschwindigkeit. Da nicht im-

mer nur Vollellipsen gezeichnet werden sollen, sondern auch Teilellipsen, sollten wir als zusätzliche Parameter einen Start- und Endwinkel zulassen:

```
SUB drawellipse(ax!,ay!,bx!,by!,mx!,my!,sw!,ew!) STATIC
'AUFGABE  :zeichnet eine Ellipse in das aktuelle Ausgabefenster
'PARAMETER:=>ax!,ay!,bx!,by! spannen die Ellipse auf
'            mx!,my! legen den Mittelpunkt der Ellipse fest
'            sw!,ew! geben den Startwinkel und den Endwinkel,
'                    ab dem (bzw. bis zu dem) gezeichnet werden
'                    soll
  SHARED maxlinien,pi2!
  alpha!=sw!
  anzlinien=ABS(sw!-ew!)*maxlinien/pi2! 'Anzahl der Linien
  IF anzlinien=0 THEN
    anzlinien=maxlinien
    beta!=p12!/maxlinien
  ELSE
    beta!=ABS(sw!-ew!)/anzlinien  'Schrittwinkel berechnen
  END IF
  x!=ax!*COS(alpha!)+bx!*SIN(alpha!)+mx! 'Startpunkt berechnen
  y!=ay!*COS(alpha!)+by!*SIN(alpha!)+my!
  CALL setpoint(x!,y!)  'Startpunkt ausgeben
  FOR i=1 TO anzlinien
    alpha!=alpha!+beta!  'neuen Winkel berechnen
    x!=ax!*COS(alpha!)+bx!*SIN(alpha!)+mx!  'neuer Punkt
    y!=ay!*COS(alpha!)+by!*SIN(alpha!)+my!
    CALL drawline(x!,y!)  'vom alten Punkt bis zum neuen
  NEXT i                  'eine Linie zeichen
END SUB
```

Am Anfang dieser Routine wird die nötige Anzahl von Linien berechnet, mit der die Ellipse gezeichnet werden soll. Wird eine Vollellipse gezeichnet, so ist diese Anzahl natürlich maximal, ist nur eine Halbellipse gewünscht, so halbiert sich diese Anzahl, usw. Liegt diese Anzahl fest, so kann nun der Schrittwert bestimmt werden, mit dem der Winkel vom Startwert bis zum Endwert kontinuierlich erhöht wird. Die eigentliche Rotation der Vektoren geschieht dann mit den beiden Funktionen Sinus und Cosinus.

Als Beispiel für die Anwendung der obigen drei Routinen, wollen wir nun eine Zeichenprozedur für einen Kegel entwickeln. Diese Prozedur soll die Projektion des Kegels in alle drei Projektionsfenster zeichnen und zum Schluß die Ausgabe wieder in das vierte Fenster, das zur textuellen Ein- und Ausgabe verwendet wird, legen. Um einen Kegel darzustellen, wird zuerst die Grundfläche ausgegeben und danach von dieser Grundfläche zur Kegelspitze vier Linien. Die Anfangspunkte dieser Linien werden durch die Grundfläche aufspannenden Vektoren festgelegt:

```
SUB drawkegel(ptr) STATIC
  SHARED k!(),pi2!
  mx!=k!(ptr,1,0)
  my!=k!(ptr,1,1)
  mz!=k!(ptr,1,2)
  ax!=k!(ptr,2,0)
  ay!=k!(ptr,2,1)
  az!=k!(ptr,2,2)
  bx!=k!(ptr,3,0)
  by!=k!(ptr,3,1)
  bz!=k!(ptr,3,2)
  cx!=k!(ptr,4,0)
  cy!=k!(ptr,4,1)
  cz!=k!(ptr,4,2)
  WINDOW OUTPUT 1
  CALL drawellipse(ax!,ay!,bx!,by!,mx!,my!,0!,pi2!)
  CALL setpoint(ax!+mx!,ay!+my!)
  CALL drawline(cx!+mx!,cy!+my!)
  CALL drawline(mx!-ax!,my!-ay!)
  CALL setpoint(bx!+mx!,by!+my!)
  CALL drawline(cx!+mx!,cy!+my!)
  CALL drawline(mx!-bx!,my!-by!)
  WINDOW OUTPUT 2
  CALL drawellipse(ax!,az!,bx!,bz!,mx!,mz!,0!,pi2!)
  CALL setpoint(ax!+mx!,az!+mz!)
  CALL drawline(cx!+mx!,cz!+mz!)
  CALL drawline(mx!-ax!,mz!-az!)
  CALL setpoint(bx!+mx!,bz!+mz!)
  CALL drawline(cx!+mx!,cz!+mz!)
  CALL drawline(mx!-bx!,mz!-bz!)
  WINDOW OUTPUT 3
  CALL drawellipse(ay!,az!,by!,bz!,my!,mz!,0!,pi2!)
  CALL setpoint(ay!+my!,az!+mz!)
```

```
    CALL drawline(cy!+my!,cz!+mz!)
    CALL drawline(my!-ay!,mz!-az!)
    CALL setpoint(by!+my!,bz!+mz!)
    CALL drawline(cy!+my!,cz!+mz!)
    CALL drawline(my!-by!,mz!-bz!)
    WINDOW OUTPUT 4
  END SUB
```

Es fehlen jetzt nur noch die Zeichenroutinen für die restlichen 10 Körper, die jedoch nicht so schwer zu entwickeln sind. Probieren Sie, verehrter Leser, es doch einmal selbst. Sollten Sie Schwierigkeiten haben, so verweise ich erneut auf den Editor, den Sie auf der Diskette finden können.

4.2.3.4 Die Datenoperationen

Jetzt kennen Sie schon viele Werkzeuge, aus denen sich ein solcher Editor zusammenbauen läßt, doch es sind bei weitem nicht alle. Stellen Sie sich mal vor, Sie hätten schon einige Körper eingegeben und wollen nun die Daten des ersten Körpers bearbeiten, oder Sie stellen fest, daß Sie irgendeinen Körper ein zweites Mal, bloß leicht verändert, benötigen und wissen die entsprechenden Daten nicht mehr, oder Sie bemerken, daß Ihre letzte Eingabe vollkommener Unsinn ist und wollen nun den gesamten Körper löschen. Bis jetzt stellt Ihnen der Editor noch nicht die entsprechenden Funktionen zur Verfügung, wir müssen sie also noch hinzufügen.

Fangen wir mit dem Schwierigeren an, dem Löschen eines Elementes. Hierfür gibt es mehere Möglichkeiten. Die erste wäre, alle Körper, die in unserem Array k! rechts von dem zu löschenden stehen, um eine Position nach links zu rücken. Diese Methode hat zum einen den Nachteil, daß der Zeitaufwand bei sehr großen Feldern nicht unerheblich ist. Zum anderen verändern sich die Nummern aller verrückten Körper, was auch nicht ideal ist, da diese Nummern der Orientierung dienen. Besser ist es, das gelöschte Element als frei zu markieren. Wird dann im Laufe der Eingabe ein freier Platz im Array k! gesucht, so kann dieses Element an der freien Stelle eingefügt werden.

Hier taucht sofort die nächste Frage auf: Wie findet unser Programm am schnellsten diese freien Plätze?

Hierzu gibt es auch eine ganz einfache Lösung. Man führt eine neue, globale Variable ein (kfree), die auf einen freien Platz zeigt. Wird also ein Körper gelöscht, so wird der Index dieses Körpers in kfree gespeichert. Was aber, wenn vorher schon ein Element gelöscht wurde? Dann geht doch die Information, wo sich dieser freie Platz befindet, unter. Man müßte sich also auch diesen Zeiger merken, und dies geschieht am einfachsten in einer der noch freien Positionen von k!(kfree,..,..), zum Beispiel in k!(kfree,0,1). Werden während des Programmablaufs mehrere Körper gelöscht, so wird allmählich eine verkettete Liste von freien Elementen aufgebaut, was zur Verdeutlichung noch einmal in der unten stehenden Abbildung gezeigt wird.

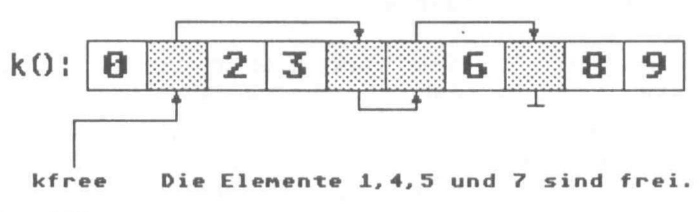


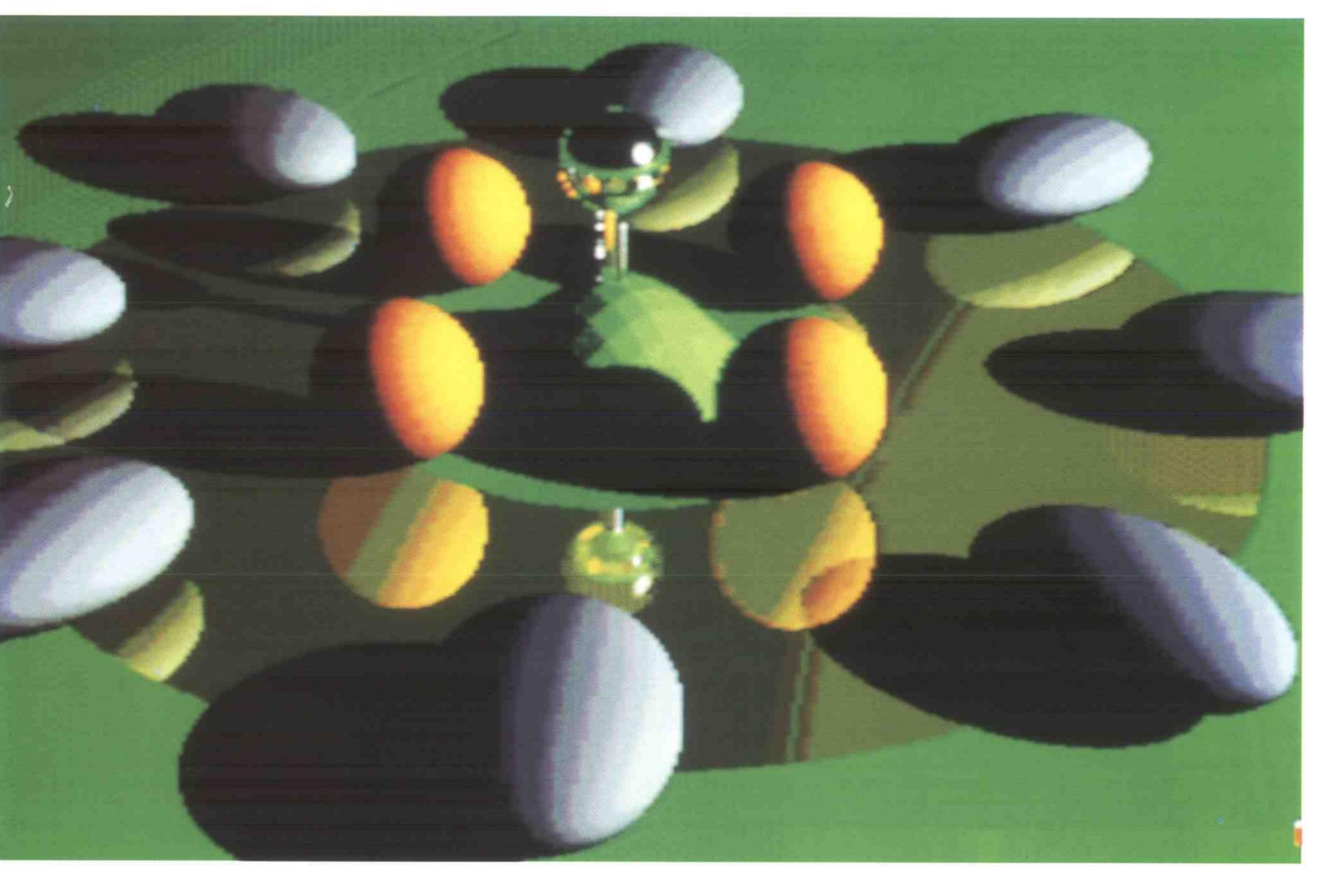
Abbildung 4.22

Diese Aufgabe, eine Liste von freien Elementen aufzubauen, übernimmt folgende Prozedur, die jedoch nicht das nullte Element löscht, da dieses als Anfangselement eine besondere Bedeutung hat:

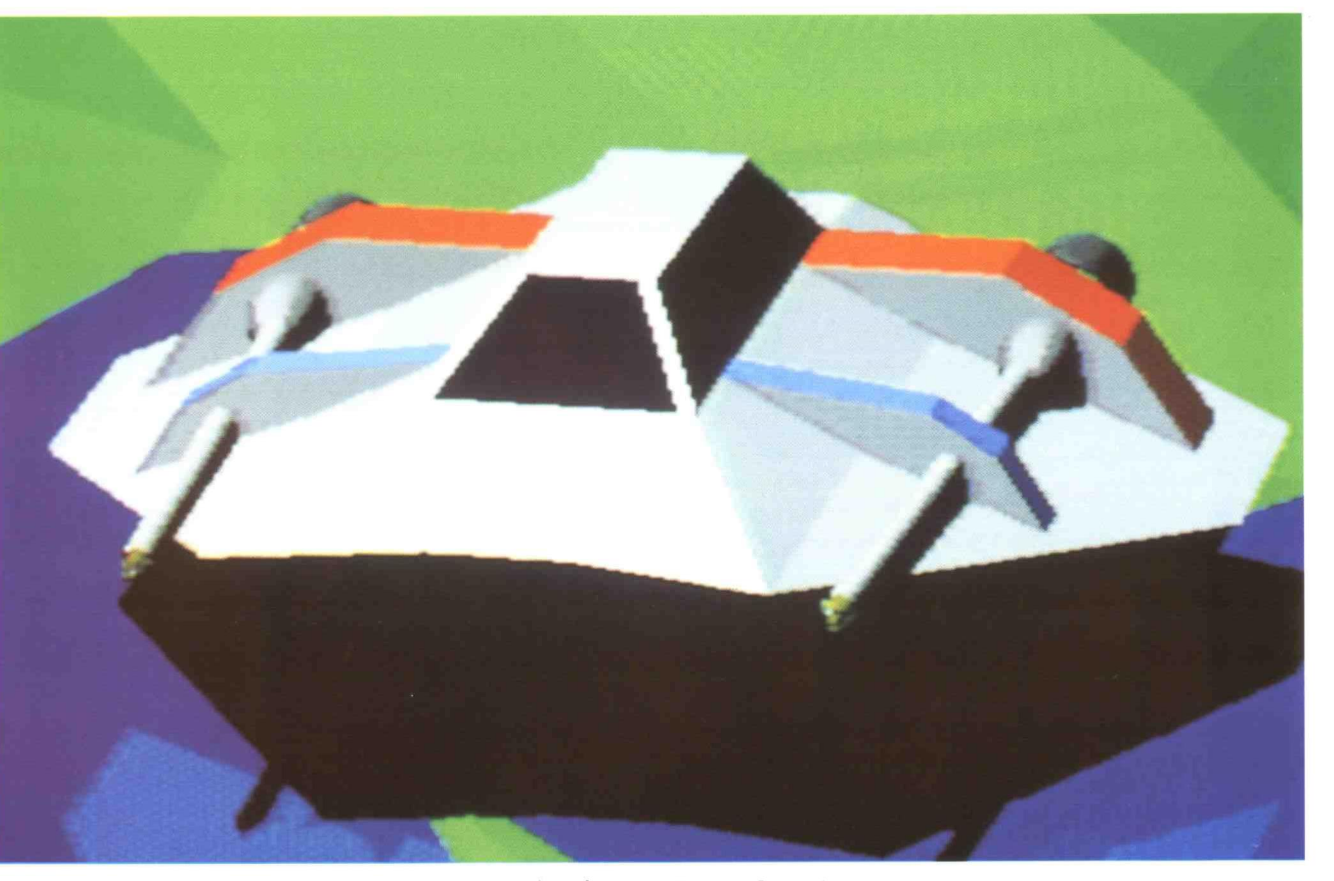
```
SUB loesche(ptr) STATIC
'AUFGABE  :löscht einen Körper
'PARAMETER:=>ptr zeigt auf den Körper
  SHARED k!(),kfree
  IF ptr><0 THEN 'nicht das Anfangselement
    FOR i=0 TO 5 'Löschen
      FOR j=0 TO 2
        k!(ptr,i,j)=0
      NEXT j
    NEXT i
```



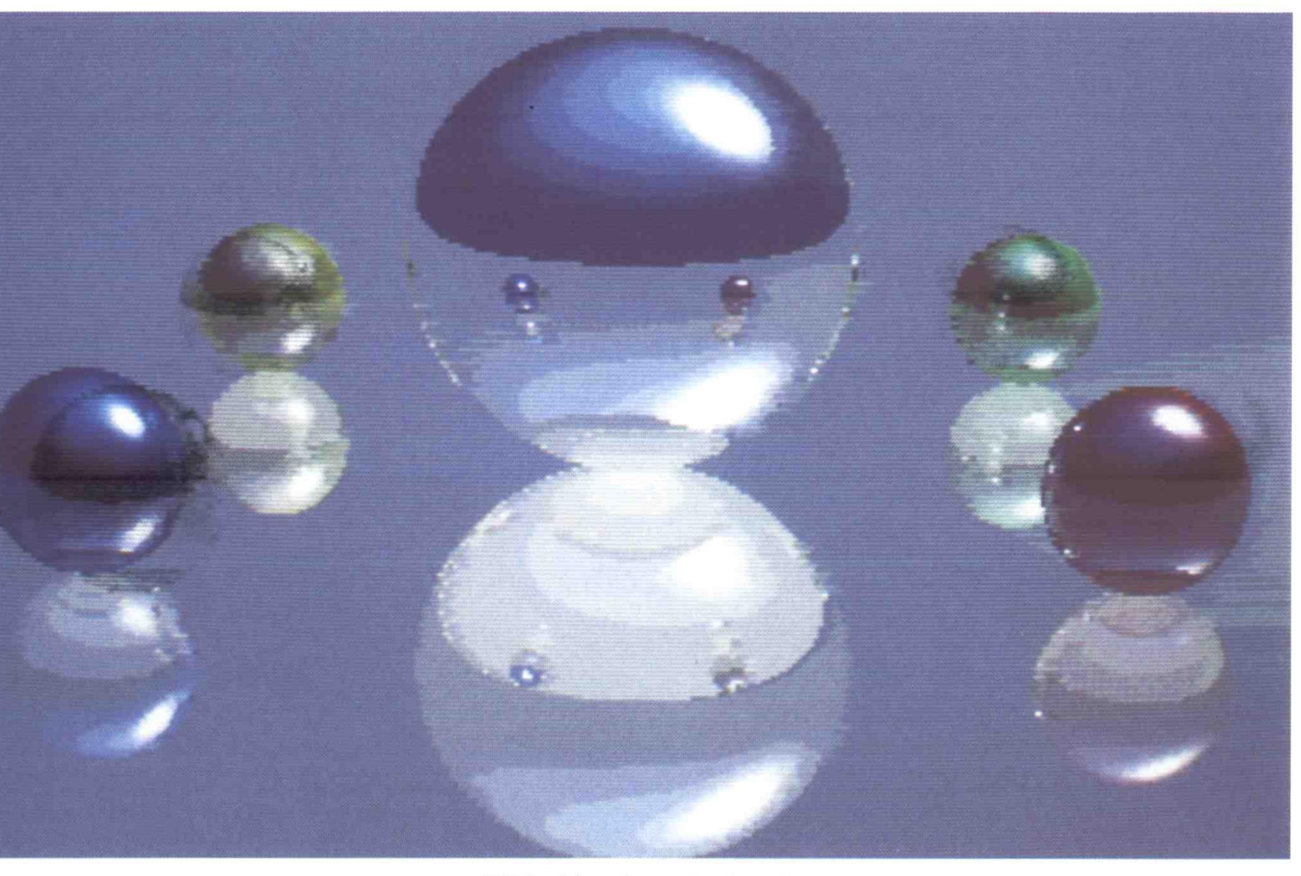
Bildschirmfoto 1: Demo



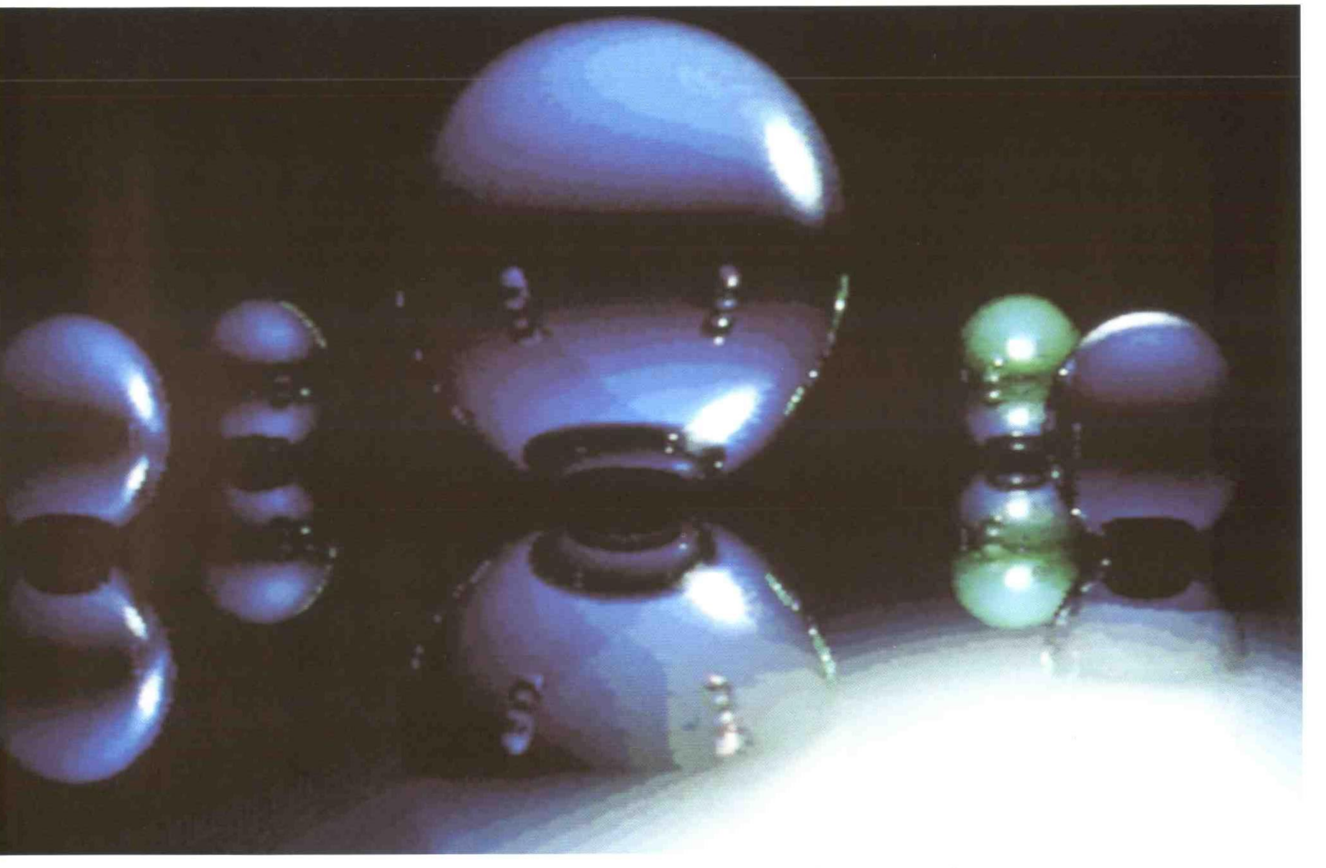
Bildschirmfoto 2: Bezier 2



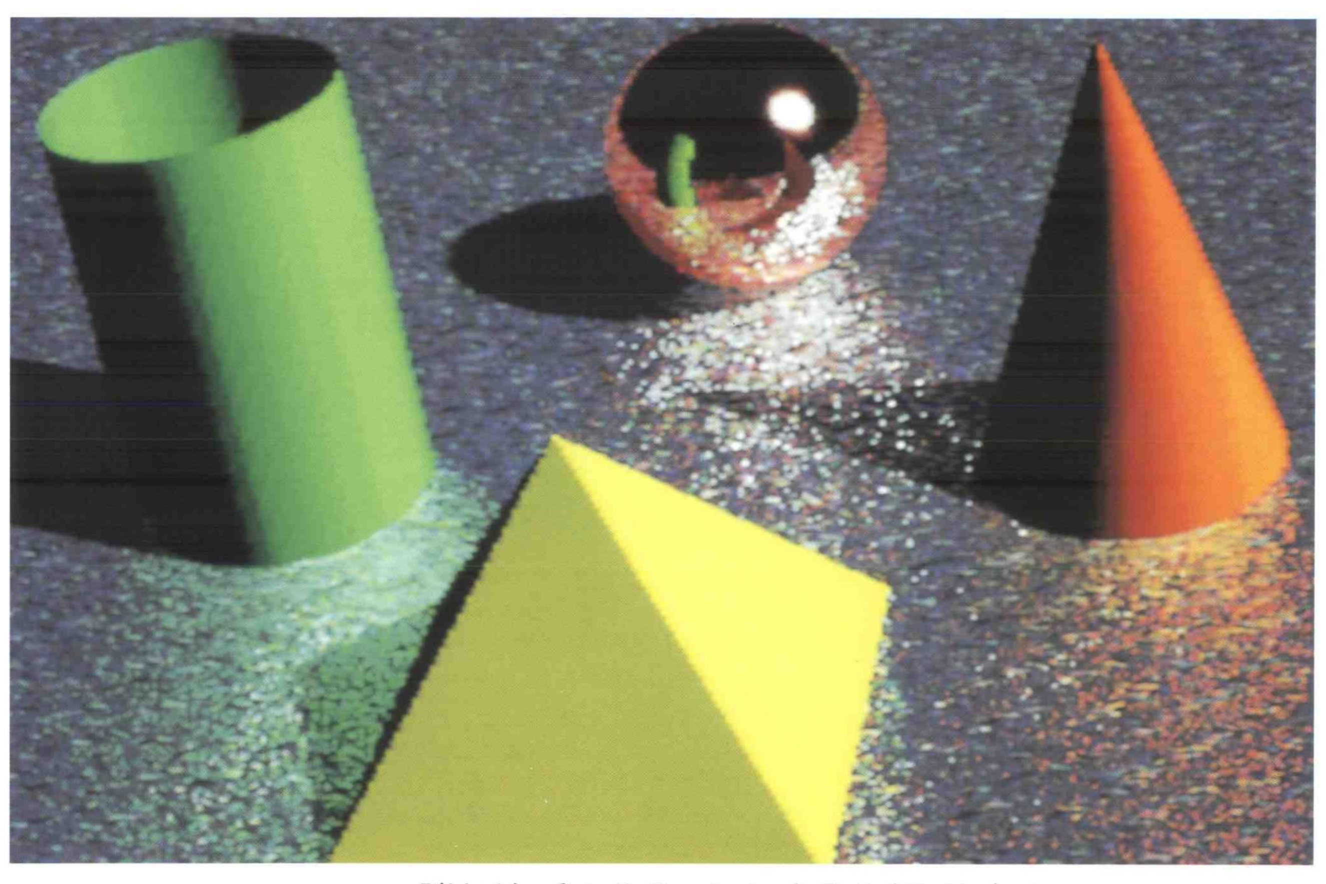
Bildschirmfoto 3: Snow-Speeder



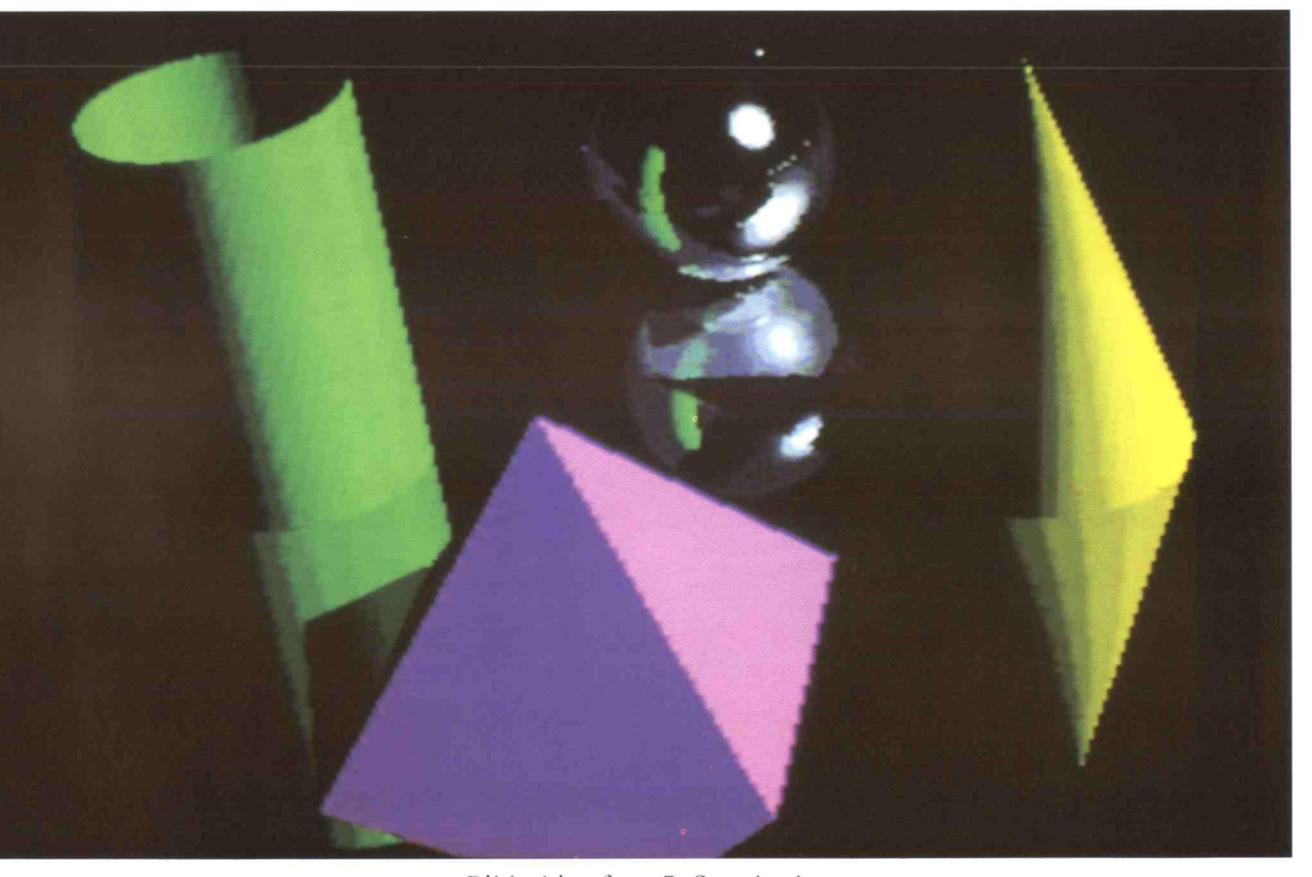
Bildschirmfoto 4: Kugel



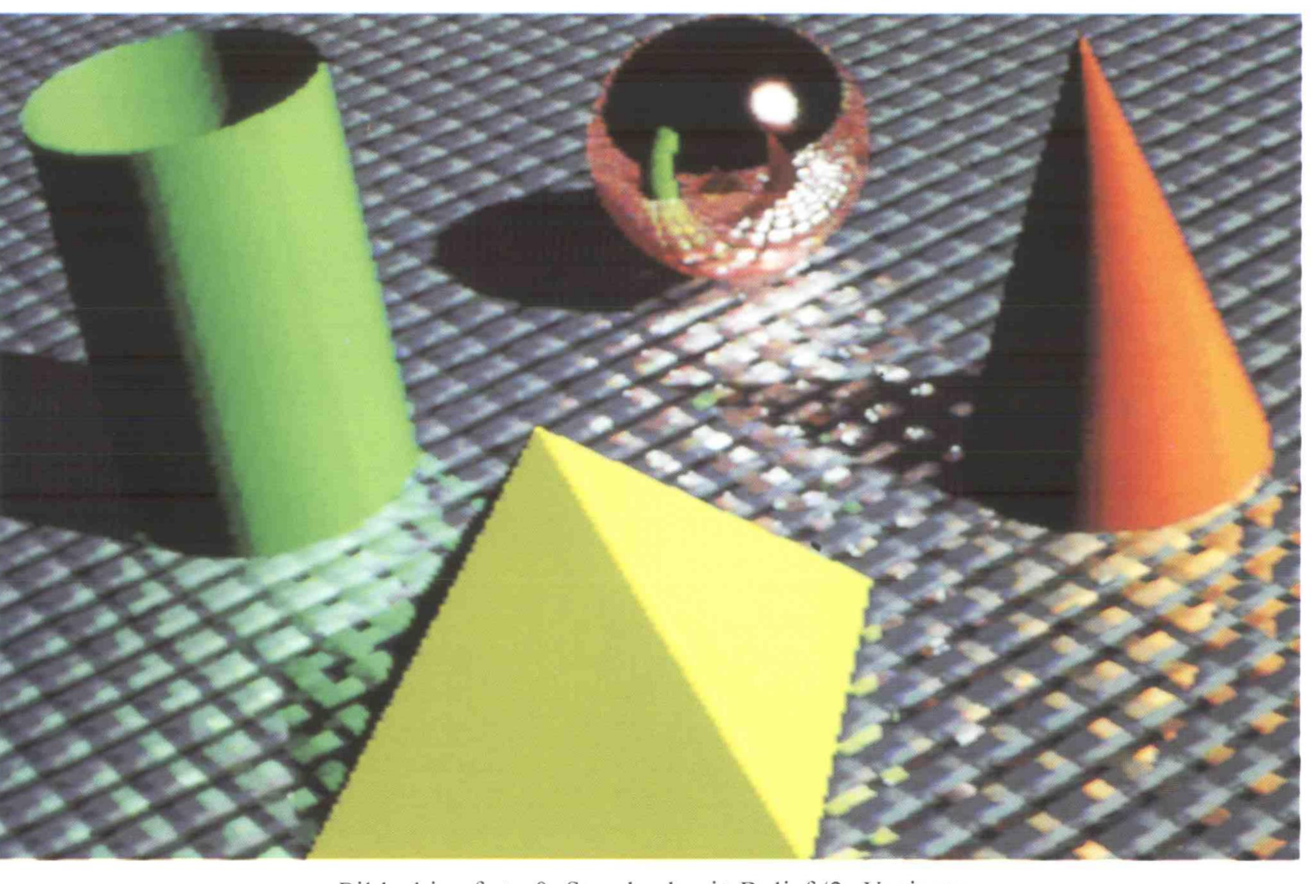
Bildschirmfoto 5: Kugel mit grauem Untergrund



Bildschirmfoto 6: Standard mit Relief/1. Variante



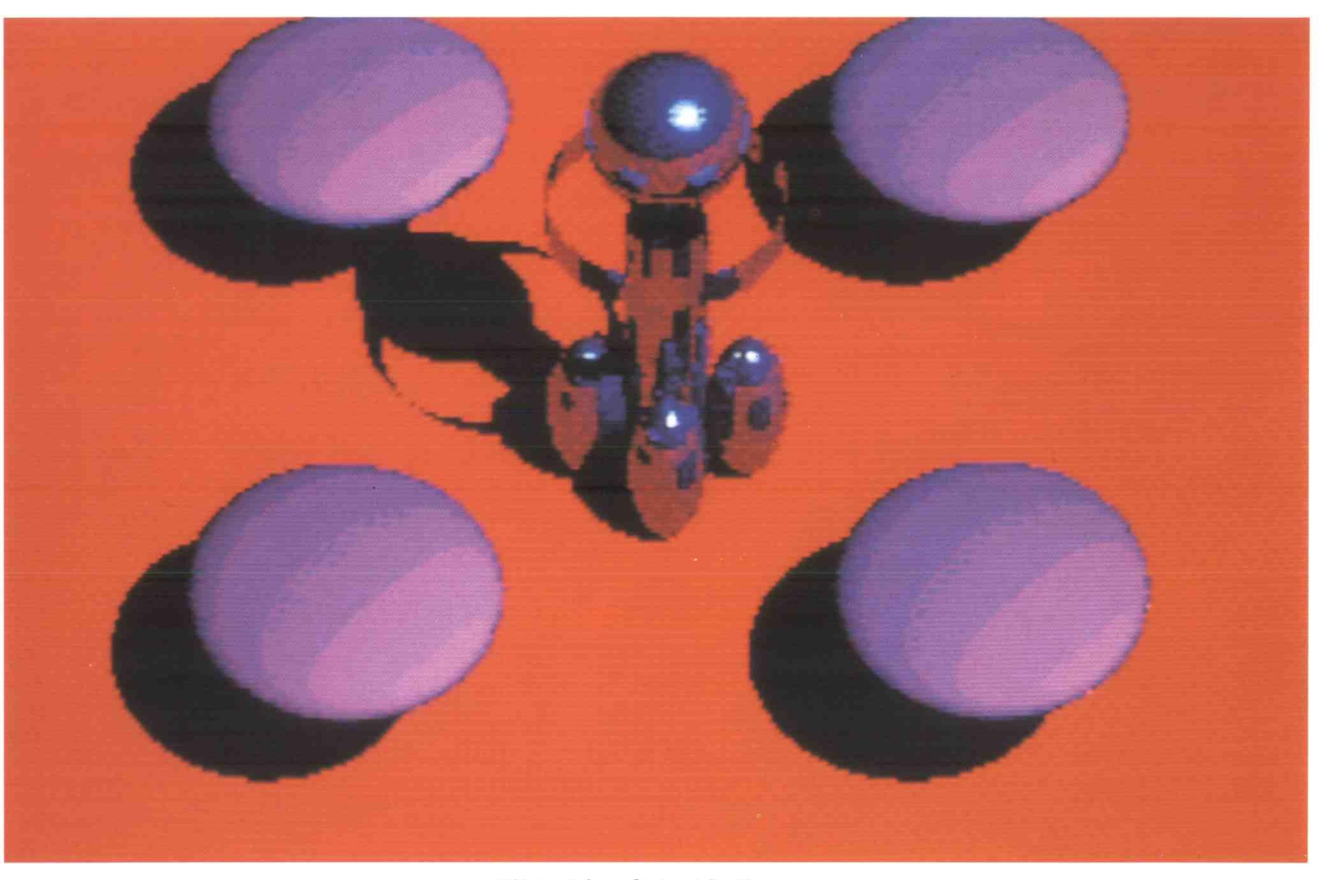
Bildschirmfoto 7: Standard



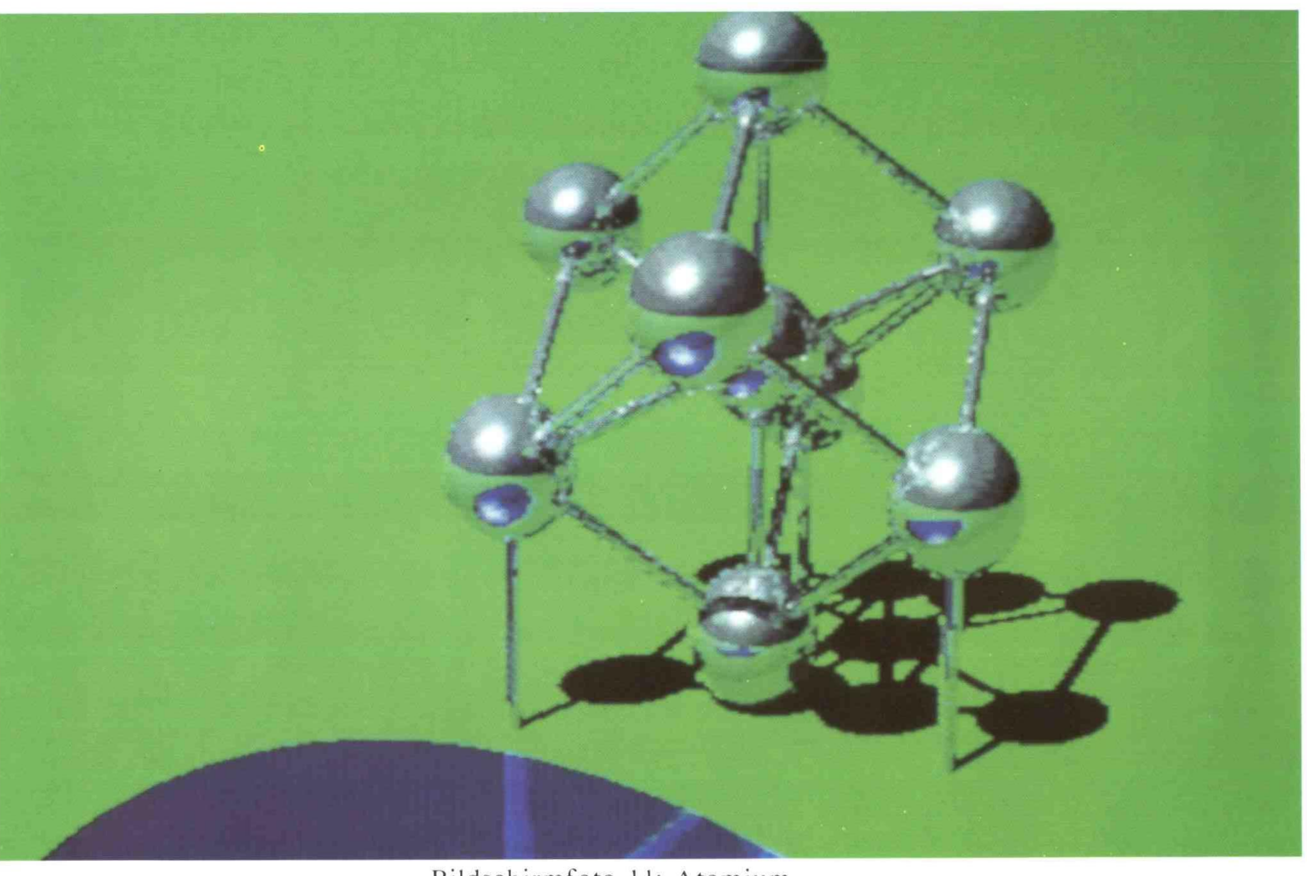
Bildschirmfoto 8: Standard mit Relief/2. Variante



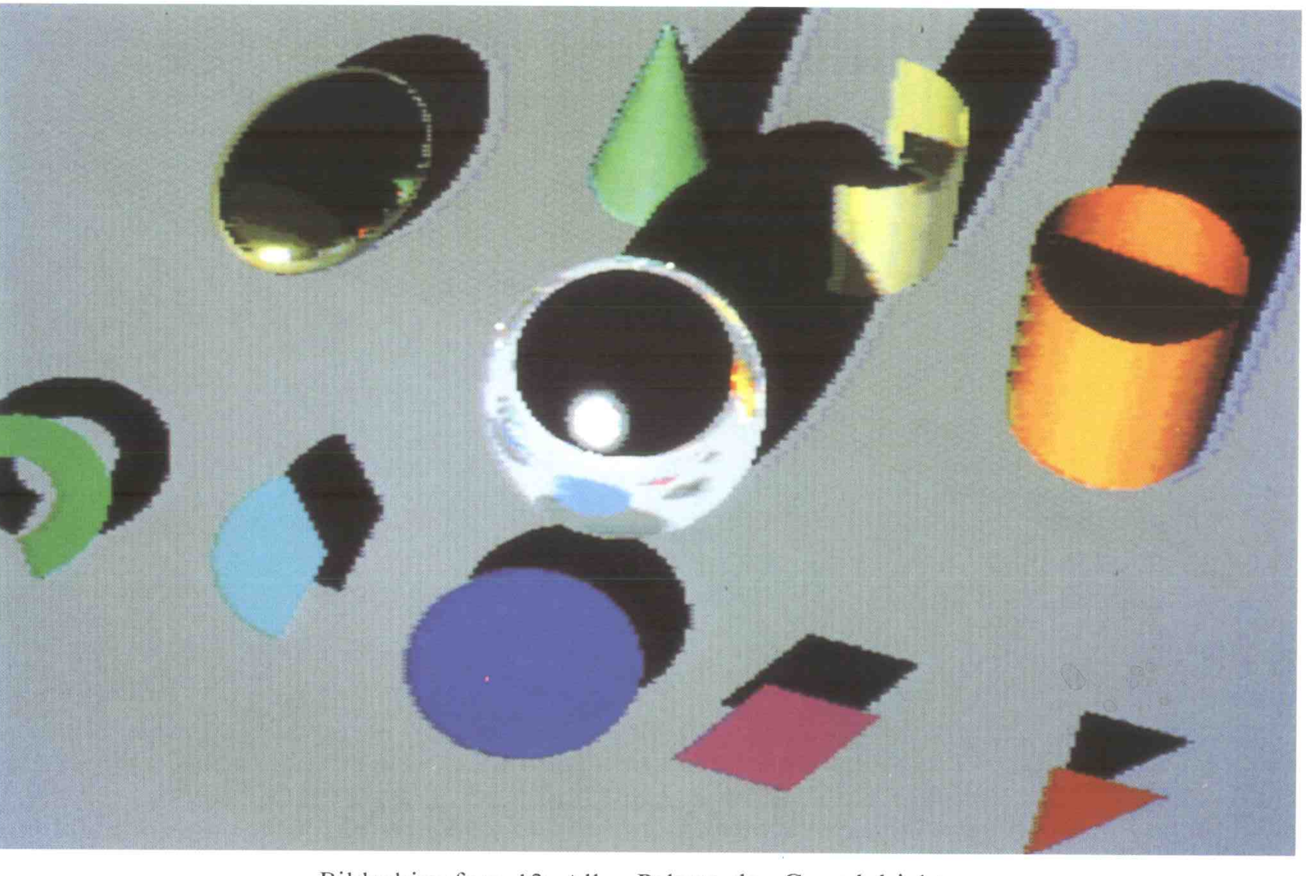
Bildschirmfoto 9: Variation zum Thema Relief



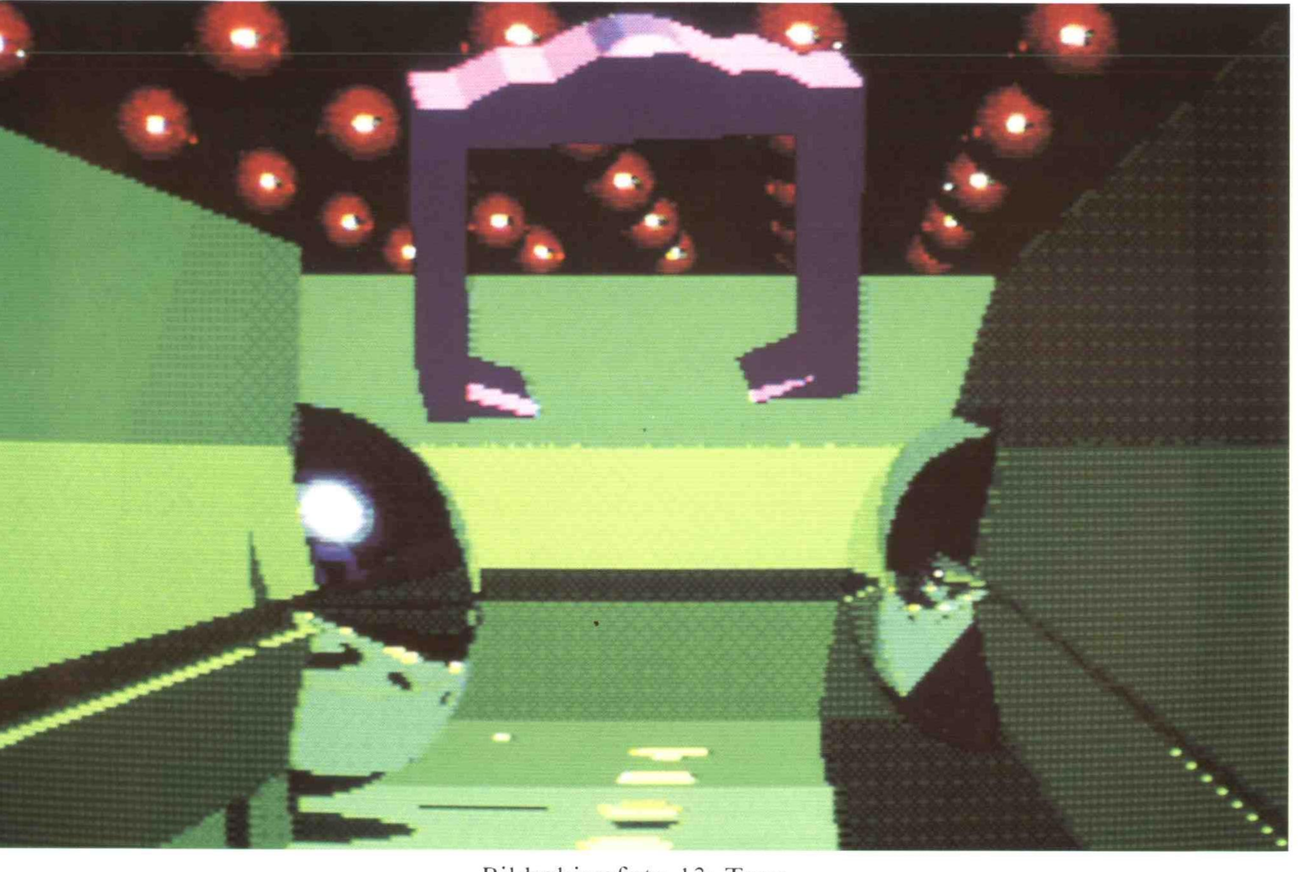
Bildschirmfoto 10: Ptower



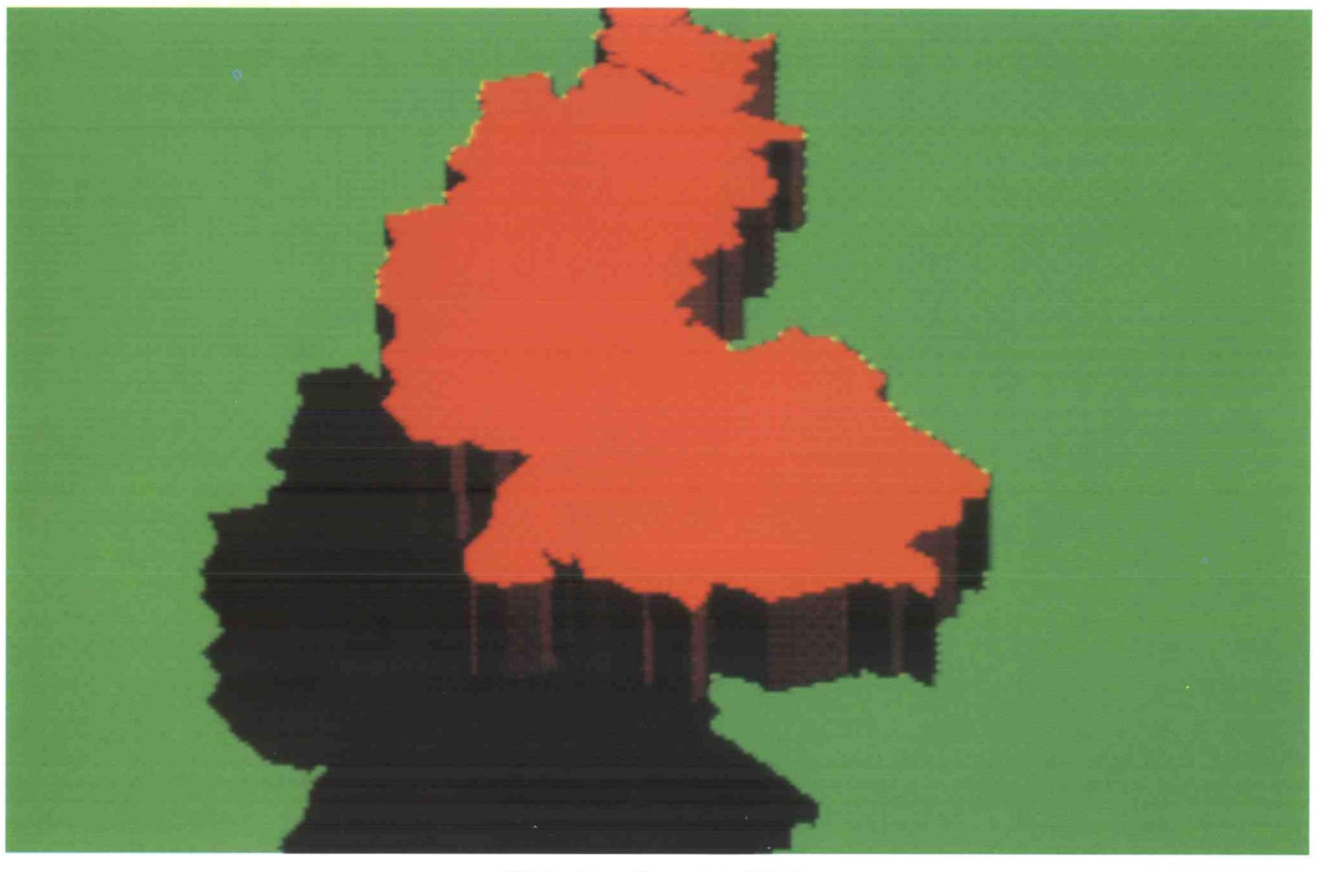
Bildschirmfoto 11: Atomium



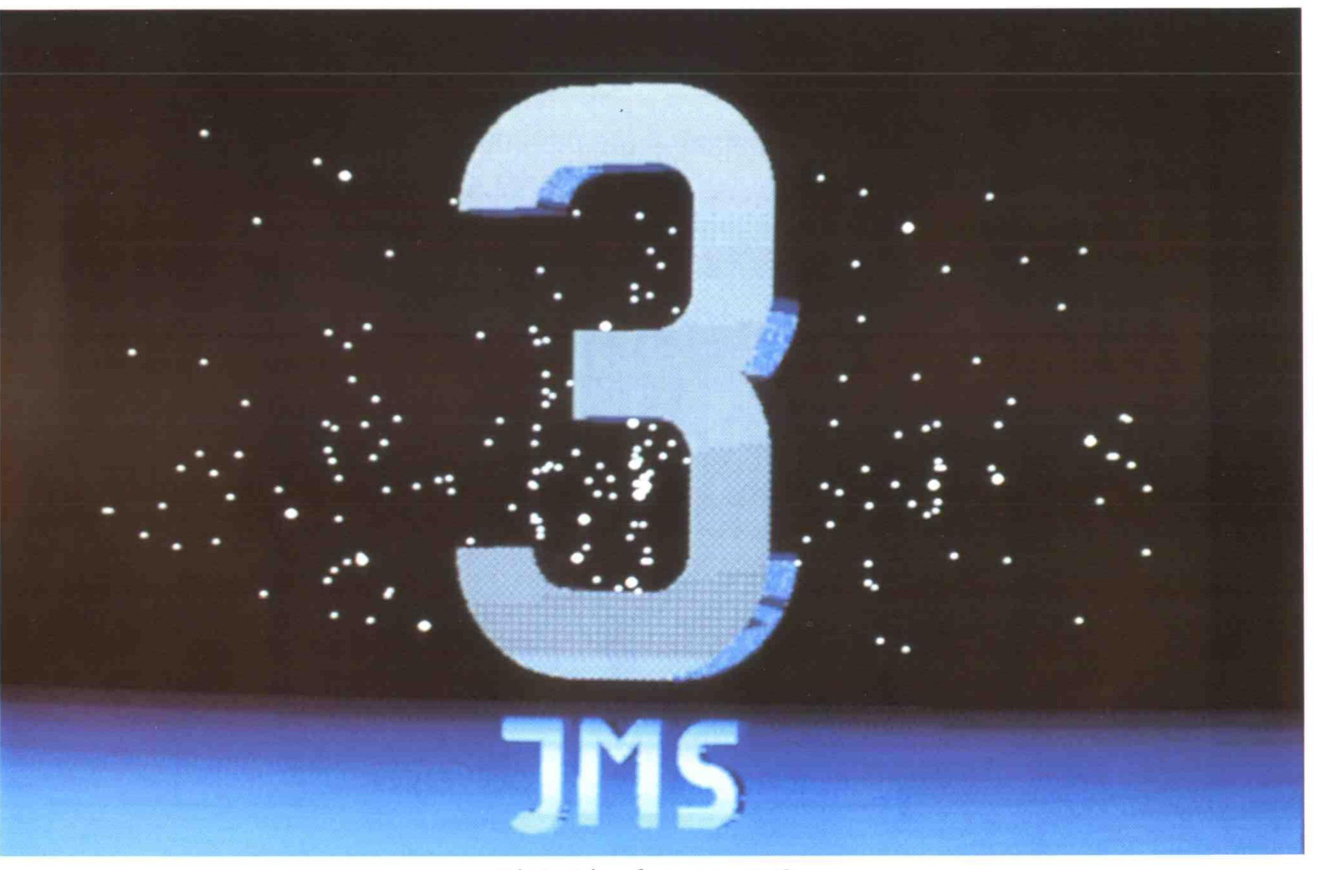
Bildschirmfoto 12: All - Palette der Grundobjekte



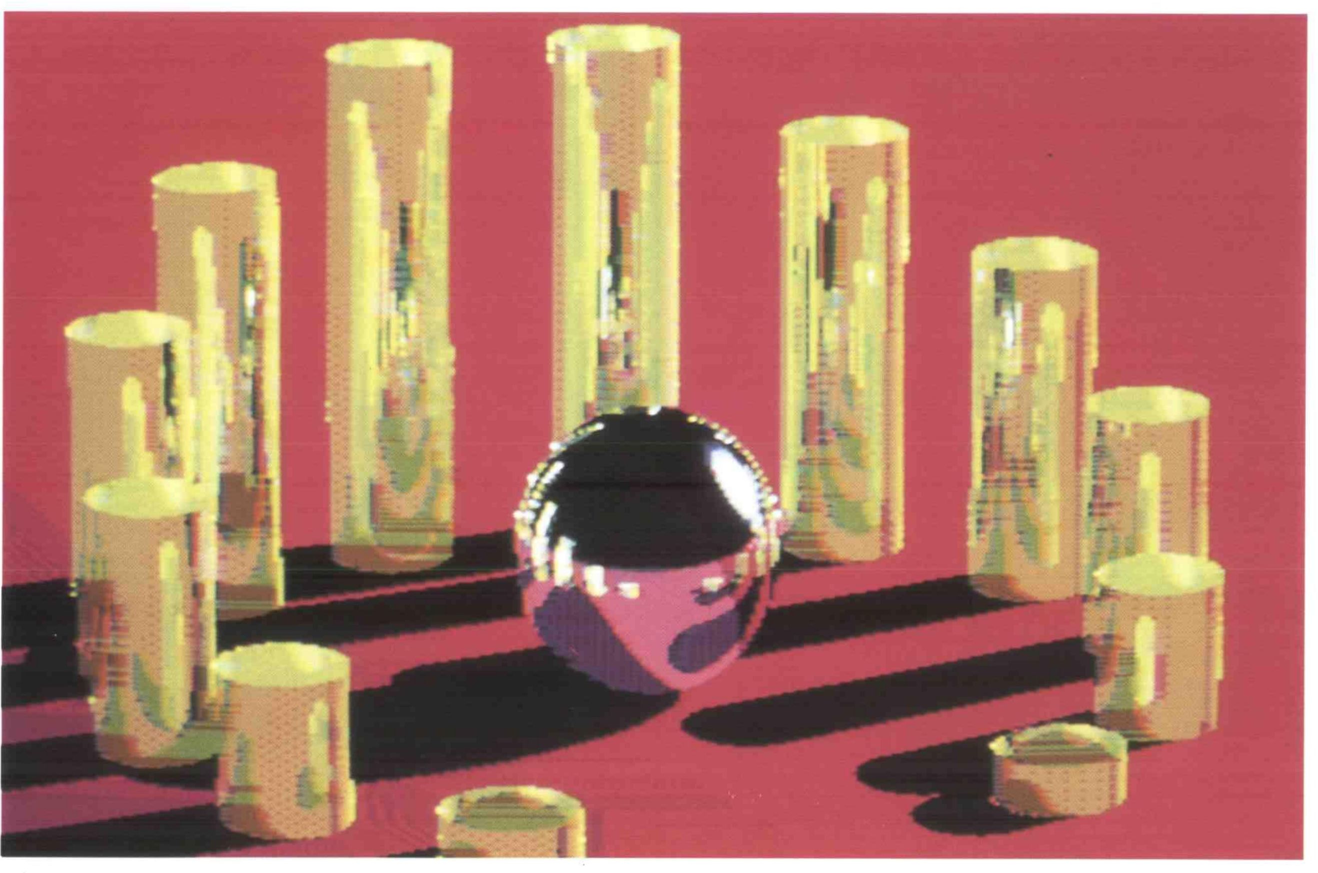
Bildschirmfoto 13: Tron



Bildschirmfoto 14: BRD



Bildschirmfoto 15: JMS



Bildschirmfoto 16: Crown

```
    k!(ptr,0,0)=-1    'Körper als frei markieren
    k!(ptr,0,1)=kfree 'und an die Freiliste anhängen
    kfree=ptr
    CALL links(ptr)
  END IF
END SUB
```

Um jetzt diesen als frei markierten Speicherplatz anzufordern, fehlt nur noch eine entsprechende Prozedur, die davon ausgeht, daß bei Programmstart das gesamte Array k! als frei markiert und an die Freiliste angehängt wurde:

```
SUB newelem(ptr) STATIC
'AUFGABE  :holt die Adresse eines freien Platzes in der Liste
'PARAMETER:<=ptr zeigt auf diese Stelle
  SHARED kfree,k!(),kanz
  IF kfree<kanz THEN
    ptr=kfree
    kfree=k!(ptr,0,1)
    k!(ptr,0,0)=0
    k!(ptr,0,1)=0
    k!(ptr,0,2)=1
  END IF
END SUB
```

Das sind nun unsere eigenen Speicherverwaltungsroutinen, die übrigens nach dem gleichen Prinzip arbeiten, wie die entsprechenden Betriebssystemprozeduren des Amigas.

Um ein Element in dem Array zu kopieren, brauchen wir uns nur noch einen Zeiger auf einen freien Speicherplatz beschaffen und Wert für Wert von einem zu anderen Element schreiben:

```
SUB copy(ptr) STATIC
'AUFGABE  :kopiert einen Körper
'PARAMETER:=> ptr zeigt auf den zu kopierenden Körper
'          <= ptr zeigt auf die Kopie
  SHARED k!()
```

```
  IF ptr><0 THEN  'nicht das Anfangselement
    old=ptr       'Zeiger retten
    CALL newelem(ptr) 'neuen Platz beschaffen
    FOR i=0 TO 5      'Kopieren
      FOR j=0 TO 2
        k!(ptr,i,j)=k!(old,i,j)
      NEXT j
    NEXT i
  END IF
END SUB
```

Weiterhin benötigen wir noch Prozeduren, um in der Liste herumwandern zu können. Wollen wir nach links, so wird der Zeiger auf das aktuelle Element um eins erniedrigt, wollen wir nach rechts, so wird er um eins erhöht. Wir müssen dabei noch berücksichtigen, daß der Zeiger nachher weder negativ ist, einen zu großen Wert enthält noch auf ein gelöschtes Element zeigt:

```
SUB links(ptr) STATIC
'AUFGABE  :holt das linke Element neben dem aktuellen
'PARAMETER:=>ptr zeigt auf den aktuellen Körper
  SHARED k!()
  IF ptr><0 THEN
    ptr=ptr-1
  END IF
  WHILE k!(ptr,0,0)=-1
    ptr=ptr-1
  WEND
  CALL showelem(ptr)
END SUB

SUB rechts(ptr) STATIC
'AUFGABE  :holt das rechte Element neben dem aktuellen
'PARAMETER:=>ptr zeigt auf den aktuellen Körper
  SHARED k!(),kanz
  old=ptr
  IF ptr<kanz THEN
    ptr=ptr+1
  END IF
  WHILE k!(ptr,0,0)=-1 AND ptr<kanz
    ptr=ptr+1
```

```
    WEND
    IF k!(ptr,0,0)=-1 THEN
      ptr=old
    ELSE
      CALL showelem(ptr)
    END IF
END SUB
```

Jetzt wollen wir uns der Datenmanipulation zuwenden, also der Vergößerung, Verschiebung und Rotation eines Körpers.

Falls die einzelnen Koordinaten eines Vektors mit ein und derselben Konstante multipliziert werden, so wird der genannte Vektor um diesen Faktor vergrößert. Falls die einzelnen Koordinaten mit verschiedenen Faktoren multipliziert werden, so wird der Vektor in die drei Grundrichtungen ungleich gestreckt, ein Effekt, den wir auch in unser Programm einbauen sollten.

Um unsere Körper zu verschieben, müssen wir lediglich ihren Bezugspunkt verschieben. Der neue Bezugspunkt kann entweder über Tastatur oder über die Maus eingegeben werden.

Komplizierter wird es mit der Rotation. Bei ihr bleibt der Bezugspunkt unberührt, es werden nur die Differenzenvektoren verändert. Diese Vektoren werden um drei Winkel im Raum nach folgender Formel gedreht:

- (x,y,z) sei der ursprüngliche Vektor

- (alpha,beta,gamma) seien die drei Winkel, um die der Vektor gedreht werden soll.

- dann ergibt sich der Rotationsvektor (X,Y,Z) mit den drei Hilfsvariablen

```
a=x*cos(gamma)-y*sin(gamma)
b=x*sin(gamma)+y*cos(gamma)
c=a*sin(beta)-z*cos(beta)
```

  aus

```
X=a*cos(beta)+z*sin(beta)
Y=c*sin(alpha)+b*cos(beta)
Z=b*sin(alpha)-c*cos(alpha)
```

Diese Formel müssen Sie mir schon so abnehmen, da ihre Herleitung einige Seiten füllen würde und auch für uns nicht weiter interessant wäre. Oft soll nicht nur ein schon eingegebener Körper rotiert werden, sondern es soll dieser Körper kopiert und dann diese Kopie rotiert werden. Deshalb bietet es sich an, dem Benutzer das Kopieren bei dieser Operation freizustellen. Unsere Rotation kann also als BASIC-Prozedur zum Beipiel folgende Form haben:

```
SUB rotation(ptr) STATIC
'AUFGABE  :rotiert einen Körper um 3 Winkel
'PARAMETER:=>ptr zeigt auf den Körper
  SHARED k!(),grad!,rad!,min!,max!
  IF ptr><0 THEN
    WINDOW 5,"Rotation",(0,129)-(314,170),4,1
    vx!=grad!*vx!  'Umwandlung nach Grad
    vy!=grad!*vy!
    vz!=grad!*vz!
    LOCATE 2,1
    PRINT"V :"
    PRINT"q<uit,d<o,c<opy&do :"
    CALL put3real(vx!,vy!,vz!,2,4,26)
    CALL get3real(vx!,vy!,vz!," ",2,4,26,min!,max!,0)
    CALL getstring(a$," ","qcd",3,21,1)
    vx!=rad!*vx!  'Umwandlung nach Rad
    vy!=rad!*vy!
    vz!=rad!*vz!
    WINDOW CLOSE 5
    WINDOW OUTPUT 4
    IF a$><"q" THEN  'Falls Quit, dann zurück
      IF a$="c" THEN  'Fall Copy, dann kopieren
        CALL copy(ptr)
      END IF
      IF k!(ptr,0,0)>=20 THEN 'Bestimme, welche Vektoren
        bis=4                 'rotiert werden müssen
      ELSE
```

```
        bis=3
      END IF
      FOR i=2 TO bis
        a!=k!(ptr,i,0)*COS(vz!)-k!(ptr,i,1)*SIN(vz!)
        b!=k!(ptr,i,0)*SIN(vz!)+k!(ptr,i,1)*COS(vz!)
        c!=a!*SIN(vy!)-k!(ptr,i,2)*COS(vy!)
        k!(ptr,i,0)=a!*COS(vy!)+k!(ptr,i,2)*SIN(vy!)
        k!(ptr,i,1)=c!*SIN(vx!)+b!*COS(vx!)
        k!(ptr,i,2)=b!*SIN(vx!)-c!*COS(vx!)
      NEXT i
      CALL showelem(ptr)
      IF k!(ptr,5,2)=1 THEN  'Fall Körper sichtbar, dann zeigen
        CALL drawelem(ptr)
      END IF
    ELSE
      WINDOW CLOSE 5
    END IF
  END IF
END SUB
```

Durch kleine Abänderungen an dieser Prozedur können Sie dann die Routinen für die Vergrößerung und Verschiebung gewinnen:

1. Bei beiden fallen die Umwandlungen von den verschiedenen Winkelsystemen weg.

2. Bei der Verschiebung wird get3real im modus -1 aufgerufen, die Abfrage, ob k!(ptr,0,0)>=20 gilt, enfällt und die FOR-TO-Schleife wird ersetzt durch:

```
k!(ptr,1,0)=vx!
k!(ptr,1,1)=vy!
k!(ptr,1,2)=vz!
```

3. Bei der Vergrößerung werden die Anweisungen in der FOR-TO-Schleife ersetzt durch:

```
k!(ptr,i,0)=vx!*k!(ptr,i,0)
k!(ptr,i,1)=vy!*k!(ptr,i,1)
k!(ptr,i,2)=vz!*k!(ptr,i,2)
```

Schon wären zwei weitere, nützliche Routinen für den Editor fertig.

4.2.3.5 Die Diskettenoperationen

Diese Operationen sind im Grunde ganz einfach. Es wird nur der Inhalt des Arrays k!() Element für Element auf die Diskette geschrieben. Es sollte bloß möglich sein, daß der Benutzer nur einen bestimmten Teil der Daten abspeichern kann, indem er die Unter- und Obergrenze der Indizes angibt. Bevor nun dieser Bereich, beziehungsweise nur die Elemente dieses Bereichs, die nicht gelöscht sind, auf die Diskette geschrieben werden, muß noch vorher die Anzahl der abzuspeichernden Elemente bestimmt und vor den ganzen Daten auf Diskette gesichert werden. Um kenntlich zu machen, daß das abgespeicherte File auf der Diskette ein Datenfile von unserem Editor ist, wird an den Filenamen noch ".list" angehängt.

```
SUB savelist STATIC
'AUFGABE:siehe entsprechende Routine fuer die Materialliste
  SHARED k!(),kanz,zeichen$,file$
  WINDOW 5,"Save List",(0,129)-(314,170),0,1
  PRINT "Name:"
  PRINT "von :1"
  PRINT "bis :"
  von!=1
  bis!=kanz
  CALL putreal(bis!,3,6,10)
  CALL getstring(file$," ",zeichen$,1,6,10)
  CALL getint(von!," ",2,6,10,1!,kanz*1!)
  CALL getint(bis!," ",3,6,10,von!,kanz*1!)
  IF file$><"" THEN
    OPEN file$+".list" FOR OUTPUT AS 1
    anzahl=0
    FOR i=von! TO bis!        'Anzahl der abzuspeichernden Elemente
                              'bestimmen
      IF k!(i,0,0) ><-1 THEN
        anzahl=anzahl+1
      END IF
    NEXT i
    PRINT #1,anzahl
    FOR ptr=von! TO bis!
```

```
        IF k!(ptr,0,0)><-1 THEN  'Leere Elemente vergessen
          FOR i=0 TO 5
            PRINT #1,k!(ptr,i,0);",";k!(ptr,i,1);",";k!(ptr,i,2)
          NEXT i
        END IF
      NEXT ptr
      CLOSE 1
    END IF
    WINDOW CLOSE 5
    WINDOW OUTPUT 4
  END SUB

  SUB loadlist STATIC
  'AUFGABE:siehe entsprechende Routine fuer die Materialliste
    SHARED k!(),kanz,ptr,zeichen$,file$
    WINDOW 5,"Load List",(0,129)-(314,150),0,1
    PRINT "Name:"
    CALL getstring(file$," ",zeichen$,1,6,10)
    IF file$><"" THEN
      OPEN file$+".list" FOR INPUT AS 1
      INPUT #1,anzahl
      WHILE NOT(EOF(1))
        CALL newelem(ptr)
        FOR i=0 TO 5
          INPUT #1,k!(ptr,i,0),k!(ptr,i,1),k!(ptr,i,2)
        NEXT i
      WEND
      CLOSE 1
    END IF
    WINDOW CLOSE 5
    WINDOW OUTPUT 4
    CALL showelem(ptr)
  END SUB
```

Bei der letzten Prozedur ist noch anzumerken, daß sie nicht überprüft, ob das sich angegebene File überhaupt auf der Diskette befindet. Sollte also ein falscher Name eingegeben werden, so kommt unweigerlich die Fehlermeldung:

FILE NOT FOUND

Das Programm stoppt, und die Zeile mit der OPEN-Anweisung wird angezeigt. Sollte Ihnen das also widerfahren, so sind Sie es selbst schuld.

4.2.3.6 Die Maus und die Gadgets

Bevor wir uns mit dem interessanteren Thema, den Gadgets, beschäftigen, will ich Ihnen noch schnell die versprochene Mausabfrage zeigen. Unsere Mausabfrage soll bei jedem Drücken der linken Maustaste durchlaufen werden und die Koordinaten der Mausposition ausgeben. Dies soll natürlich nur geschehen, falls das angewählte Fenster ein Projektionsfenster ist. Ist es aber unser Ein- und Ausgabefenster, so sollte nichts geschehen. Im anderen Fall werden mittels MOUSE(3) und MOUSE(4) die x- bzw. y-Koordinate der Mausposition bestimmt. Diese Koordinaten stellen jedoch Bildschirmkoordinaten dar und nicht Raumkoordinaten, die uns aber interessieren. Sie müssen also umgerechnet werden, wobei Ausschnitt, Vergößerungsfaktor und x-Korrekturfaktor berücksichtigt werden müssen. Um die angeklickte Position zu markieren, wird am besten zur Kontrolle und Übersicht noch ein kleines Kreuz auf den Bildschirm gezeichnet. Anschließend müssen noch die Mauskoordinaten ausgegeben werden:

```
klick:
  dummy=MOUSE(0)
  x!=MOUSE(3)/(faktor!*xkorrfak!)  'Mausposition holen und umrechnen
  y!=MOUSE(4)/faktor!
  n=WINDOW(0)  'angeklicktes Fenster
  oldfenster=WINDOW(1) 'Ausgabefenster
  CALL activatewindow&(WINDOW(7))
  IF n<4 THEN  'wurde eines der Projektionsfenster angeklickt?
    WINDOW OUTPUT n
    LINE (MOUSE(3)-1,MOUSE(4))-(MOUSE(3)+1,MOUSE(4)),3  'Kreuz zeichnen
    LINE (MOUSE(3),MOUSE(4)-1)-(MOUSE(3),MOUSE(4)+1),3
    IF n=1 THEN    'Ausrechnen der Raumkoordinaten
      mox!=mx!+x!
      moy!=my!-y!
    ELSE
```

```
      IF n=2 THEN
        mox!=mx!+x!
        moz!=mz!-y!
      ELSE
        IF n=3 THEN
          moy!=my!-x!
          moz!=mz!-y!
        END IF
      END IF
    END IF
    mox!=FIX(mox!+.5) 'runden
    moy!=FIX(moy!+.5)
    moz!=FIX(moz!+.5)
    WINDOW OUTPUT 4
    mo$=STR$(mox!)+","+STR$(moy!)+","+STR$(moz!)
    LOCATE 10,8       'Löschen der alten Mauswerte und Ausgabe der neuen
    PRINT SPACE$(26)
    LOCATE 10,8
    PRINT MID$(mo$,1,26)
    WINDOW OUTPUT oldfenster
  ELSE
    CALL checkgadget  'wurde ein Gadget angeklickt?
  END IF
RETURN
```

Diese Unterprogramm muß dann nur noch mit "ON MOUSE GOSUB klick" in unser Programm eingebunden werden.

Falls Sie sich dieses Unterprogramm sorgfältig angeschaut haben, so sollte Ihnen der Aufruf von checkgadget aufgefallen sein, womit wir schon beim nächsten Punkt wären, der Implementierung von Gadgets. Ich habe hierfür folgenden Weg gewählt.

Mit der Prozedur initgadget können Sie festlegen, an welcher Position auf dem Bildschirm welches Gadget in welcher Höhe und Breite zusehen sein wird und welchen Zustand es anfangs haben soll. All diese Informationen werden in einem speziellen, globalen Array abgespeichert. Wird nun die linke Maustaste gedrückt, so wird, wie Sie es ja oben sehen konnten, die Routine checkgadget aufgerufen. Sie überprüft nun, ob der Mauszeiger innerhalb des Gadgets liegt. Sollte dies der Fall sein, so müssen folgende zwei Möglichkeiten unterschieden werden:

1. Falls das Gadget ein Regler ist, so muß der alte Regler gelöscht und an der neuen Position hingezeichnet werden. Damit unser Programm überhaupt mitbekommt, daß sich inzwischen etwas getan hat, wird die jetzige Position des Reglers im Array gadget% an der Stelle vier abgespeichert.

2. Falls das Gadget ein Symbol ist, so wird dieses Symbol, dessen Bildinformationen im Array gadgets% stehen, entweder invertiert, oder es wird ein ganz neues Symbol ausgegeben. Diese Bildinformationen wurden mittles GET am Programmstart eingeladen und werden mittels PUT wieder ausgegeben. Wieder wird der Zustand des Gadgets in gadget%(4) zwischengespeichert und ist dort vom Programm aus abrufbar.

Eine weitere Routine sorgt schließlich dafür, daß die Gadgets, die Sie nicht mehr benötigen, im Array jedoch nicht auf dem Bildschirm gelöscht werden. Letzteres müssen Sie dann schon selbst übernehmen.

```
SUB checkgadget STATIC
'AUFGABE:ueberprueft ob ein Gadget angeklickt wurde
  SHARED gadget%(),gadgets%()
  dummy=MOUSE(0)
  ax=MOUSE(3) 'Mausposition holen
  ay=MOUSE(4)
  i=0
  WHILE i<=10  'durchsuche das ganze Array
    x%=gadget%(i,0)     'die Werte aus dem Array holen
    y%=gadget%(i,1)
    h%=gadget%(i,2)
    l%=gadget%(i,3)
    wert%=gadget%(i,4)
    wind%=gadget%(i,5)
    ja%=gadget%(i,6)
    nein%=gadget%(i,7)  'Maus ueber Gadget ?
    IF x%<ax AND x%+l%>ax AND y%<ay AND y%+h%>ay AND x%>=0
       AND WINDOW(0)=wind% THEN
      old=WINDOW(1) 'Ausgabefenster retten
      WINDOW OUTPUT wind%   'alten Regler löschen,neuen setzen
      IF ja%=-1 THEN
        LINE (wert%+x%+1,y%+1)-(wert%+x%+1,y%+h%-1),0
        wert%=ax-x%-1
```

```
          LINE (wert%+x%+1,y%+1)-(wert%+x%+1,y%+h%-1),3
          gadget%(i,4)=wert%  'neuen Wert retten
        ELSE
          IF nein%=-1 THEN
            IF wert%=1 THEN  'zum ersten Mal?,dann invertieren
              gadget%(i,4)=0
              LINE (x%,y%)-(x%+l%,y%+h%),0,bf
              PUT (x%,y%),gadgets%(ja%),PRESET
            ELSE
              gadget%(i,4)=1  'sonst zurueck stellen
              LINE (x%,y%)-(x%+l%,y%+h%),0,bf
              PUT (x%,y%),gadgets%(ja%)
            END IF
          ELSE
            IF wert%=1 THEN  'zum ersten Mal?,dann nein-Bild
              gadget%(i,4)=0                         'ausgeben
              LINE (x%,y%)-(x%+l%,y%+h%),0,bf
              PUT (x%,y%),gadgets%(nein%)
            ELSE
              gadget%(i,4)=1 'sonst wieder ja-Bild ausgeben
              LINE (x%,y%)-(x%+l%,y%+h%),0,bf
              PUT (x%,y%),gadgets%(ja%)
            END IF
          END IF
        END IF
        WINDOW OUTPUT old   'altes Ausgabefenster aktivieren
        i=10
      END IF
      i=i+1
    WEND
  END SUB

  SUB initgadget(x%,y%,h%,l%,wert%,wind%,nr%,ja%,nein%) STATIC
  'AUFGABE  :legt ein neues Gadgets im Array gadget%() an
  'PARAMETER:=>x%,y% Position der linken unteren Ecke
  '            h%,l% Höhe und Länge des Rahmens
  '            wert% Startwert
  '            wind% Ausgabefenster
  '            ja%,nein% falls ja%=-1,dann Regler
  '                      falls nein%=-1,dann wird das Gadget
  '                      nach jedem Anklicken invertiert
  '                      sonst wird ein anderes Bild ausgegeben
  '                      ja% und nein% geben dann die Indizes
  '                      der graphischen Informationen im
  '                      array gadgets an
  '          <=nr% Position des Gadgets im Array
```

```
    SHARED gadget%(),gadgets%()
    IF x%>=0 AND y%>=0 AND h%>=2 AND l%>=3 AND wert%>=0
      AND wert%<l% AND wind%>0 THEN
      MOUSE OFF
      old=WINDOW(1)
      WINDOW OUTPUT wind%
      IF ja%=-1 THEN
        LINE  (x%,y%)-(x%,y%+h%)
        LINE -(x%+l%,y%+h%)
        LINE -(x%+l%,y%)
        LINE -(x%,y%)
        LINE (wert%+x%+1,y%+1)-(wert%+x%+1,y%+h%-1),3
      ELSE
        IF wert%=1 THEN
          PUT (x%,y%),gadgets%(ja%)
        ELSE
          PUT (x%,y%),gadgets%(nein%)
        END IF
      END IF
      nr%=0
      WHILE gadget%(nr%,0)><-1 AND nr%<=9
        nr%=nr%+1
      WEND
      IF gadget%(nr%,0)=-1 THEN
        gadget%(nr%,0)=x%
        gadget%(nr%,1)=y%
        gadget%(nr%,2)=h%
        gadget%(nr%,3)=l%
        gadget%(nr%,4)=wert%
        gadget%(nr%,5)=wind%
        gadget%(nr%,6)=ja%
        gadget%(nr%,7)=nein%
      END IF
      MOUSE ON
      WINDOW OUTPUT old
    END IF
  END SUB

  SUB deletegadget(i) STATIC
  'AUFGABE  :löscht ein Gadget
  'PARAMETER:=>i Index des Gadgets im Array
    SHARED gadget%()
    IF i>=0 AND i<=10 THEN
      gadget%(i,0)=-1
    END IF
  END SUB
```

Als Beispiel für die Anwendung dieser Prozeduren will ich Ihnen noch schnell zwei Routinen des Materialeditors vorgestellen. In diesem Editor müssen nur fünf Realwerte eingelesen werden, und zwar ein Rot-, Grün-, Blauwert, ein Wert, der die Helligkeit im Schatten bestimmt, und ein Spiegelungsfaktor. All diese Werte müssen zwischen eins und null liegen. Da durch die Rot-, Grün-, Blauwerte eine Farbe festgelegt werden soll, ist es natürlich sehr hilfreich, wenn der Benutzer diese Farbe sofort auf dem Bildschirm sehen kann, was sich aber leicht durch die PALETTE 2,..,..,.. Anweisung einbauen läßt. Hier nun also die Routine, um ein Materialelement anzeigen zu lassen (das Gegenstück zu showelem):

```
SUB showmat(matptr) STATIC
'AUFGABE:siehe entsprechende Routine fuer die Körperliste
  SHARED mat!(),nr1%,nr2%,nr3%,nr4%,nr5%,nr6%
  CLS
  PRINT "Material:"
  CALL putreal(matptr*1!,1,10,10)
  IF matptr ><0 THEN
    LOCATE 1,20
    PRINT "R+G+B="
    PALETTE 2,mat!(matptr,0),mat!(matptr,1),mat!(matptr,2)
    LINE (250,0)-(260,7),2,bf
    PRINT "Rot       :"
    PRINT
    PRINT "GrÜn      :"
    PRINT
    PRINT "Blau      :"
    PRINT
    PRINT "Schatten  :"
    PRINT
    PRINT "Spiegelung:"
    w1%=mat!(matptr,0)*101
    w2%=mat!(matptr,1)*101
    w3%=mat!(matptr,2)*101
    w4%=mat!(matptr,3)*101
    w5%=mat!(matptr,4)*101
```

```
      CALL initgadget(110,8,10,102,w1%,5,nr1%,-1,-1)
      CALL initgadget(110,26,10,102,w2%,5,nr2%,-1,-1)
      CALL initgadget(110,44,10,102,w3%,5,nr3%,-1,-1)
      CALL initgadget(110,62,10,102,w4%,5,nr4%,-1,-1)
      CALL initgadget(110,80,10,102,w5%,5,nr5%,-1,-1)
      CALL initgadget(110,96,15,30,1,5,nr6%,0,-1)
    END IF
  END SUB
```

Bei dem letzten initgadget-Aufruf ist übrigens die Null die Adresse des OK-Symbols, das angeklickt werden muß, falls die Dateneingabe beendet werden soll.

Genauso einfach wie die obige Prozedur wird nun mit unseren Gadgets die Routine zum Einlesen der Materialdaten. Sie wird sofort nach showmat aufgerufen:

```
SUB getmat(matptr) STATIC
'AUFGABE:siehe entsprechende Routine fuer die Körperliste
  SHARED mat!(),gadget%(),nr1%,nr2%,nr3%,nr4%,nr5%,nr6%
  CALL showmat(matptr)
  WHILE gadget%(nr6%,4)><0
    PALETTE 2,gadget%(nr1%,4)/100,gadget%(nr2%,4)/100,gadget%
    (nr3%,4)/100
  WEND
  mat!(matptr,0)=gadget%(nr1%,4)/101
  mat!(matptr,1)=gadget%(nr2%,4)/101
  mat!(matptr,2)=gadget%(nr3%,4)/101
  mat!(matptr,3)=gadget%(nr4%,4)/101
  mat!(matptr,4)=gadget%(nr5%,4)/101
  CALL deletegadget(nr1%)
  CALL deletegadget(nr2%)
  CALL deletegadget(nr3%)
  CALL deletegadget(nr4%)
  CALL deletegadget(nr5%)
  CALL deletegadget(nr6%)
END SUB
```

Das waren nun auch die letzten wichtigen Routinen, die zusammengewürfelt einen Editor für den Ray-Tracer ergeben können. Letzte Tips, was Sie hierbei zu beachten haben, finden sich im folgenden Kapitel.

4.2.4 Wir bauen uns einen Editor

Was ich Ihnen bisher vorgestellt habe, ist im Grunde nur eine Loseprozedursammlung. Um hieraus ein vollständiges Programm zu erhalten, müssen noch zusätzliche Funktionen hinzugefügt werden. Ein Problem ist zum Beispiel, wie diese Funktionen dem Benutzer zur Verfügung gestellt werden. Sollen wie im CLI einzelne Befehle dafür sorgen, daß nun der Materialeditor läuft, der Bildschirm neu aufgebaut oder der Ausschnitt verschoben wird? Damit dem Benutzer viel unnötige Tipparbeit erspart bleibt, ist es vielleicht besser, alle Funktionen über ein Menü abrufbar zu haben und dies noch zugleich mit der ersten Variante zu kombinieren, indem einige wichtige Funktionen über einen Tastendruck aktiviert werden können. Die hierzu nötigen Prozeduren sind sehr einfach und bestehen im Grunde nur aus einer Reihe von IF-Abfragen.

Als nächstes müssen sämtliche globalen Variablen initialisiert und globale Felder dimensioniert werden. Ebenfalls sind ein Screen und vier Windows zu öffnen, wobei das textuelle Ein- und Ausgabefenster zum Programmstart mit einem Anfangsbild versehen werden muß. Denken Sie auch daran, die Arrays k! und mat! am Anfang zu verketten, also alle Elemente als frei zu deklarieren und in die Freiliste einzuhängen.

Nützlich ist es übrigens, wenn ein Sprung von dem aktuellen Körperelement auf ein beliebiges möglich ist. So brauchen Sie sich nicht von Element zu Element zu hangeln und ersparen sich somit viel Zeit. Und noch eine Funktion sollte erwähnt werden, und zwar die Show-Funktion. Sie hebt den aktuellen Körper auf den Projektionsfenstern optisch hervor (d.h. mit einer anderen Farbe). Bei großen Datenbeständen verlieren Sie so nicht den Überblick.

Wichtig ist schließlich auch eine Möglichkeit, Ihr Programm verlassen zu können, ohne gleich einen Reset durchführen zu müssen. Bei dieser Quit-Prozedur sollten Sie noch eine Abfrage einbauen, ob der Benutzer wirklich das Programm beenden möchte oder ob es sich nur um ein Versehen handelt. Hier bietet sich natürlich wieder die Verwendung eines Gadgets an.

So, nun genug der Worte. Lassen Sie Taten folgen. Viel Spaß beim Programmieren.

4.3 Das Hauptprogramm

Sie haben sicher schon gemerkt, daß wir ein bestimmtes Ziel verfolgen. Alle bisher aufgelisteten Routinen können zu einem Programm zusammengefaßt werden. Dabei stellt jedoch der Objekteditor ein eigenständiges Modul dar, daß von unserem Hauptprogramm aufgerufen wird.

4.3.1 Die restlichen Routinen des Tracers

Doch kommen wir jetzt dazu, welche Routinen noch benötigt werden, damit unser später zusammengesetztes Hauptprogramm komplett ist.

Zunächst wollen wir die 'RasterInit'-Routine etwas näher beschreiben. In dieser Routine werden nämlich Benutzer- und Display-Screen aufgebaut. Im Benutzer-Screen werden später alle Eingaben vorgenommen. Im Display-Screen werden sowohl das Drahtmodell als auch das schattierte Bild dargestellt. So ist es möglich, daß der Tracer in allen Auflösungen funktioniert und trotzdem eine übersichtliche Benutzerschnittstelle (mit 60/80 Zeichen pro Zeile) geschaffen werden konnte.

```
RasterInit:
  ' Auflösung und Darstellungsmodus wählen
  WINDOW CLOSE 2 ' evt, vorhandene Windows und
  SCREEN CLOSE 1 ' Screens löschen
```

```
WINDOW 1,"", (0,0)-(631,185),6  ' neues Window mit Status-Zeile

'PALETTE 0,0,0,0                ' Farben fuer Benutzer-Screen
'PALETTE 1,1,1,1
'PALETTE 2,.4,.4,1

NWBase& = WINDOW(7)
NRastPort& = WINDOW(8)
NWScreen& = PEEKL(NWBase&+46)
' Basisadresse der Window-Struktur
' Daraus wird Adresse der Screen-Struktur extrahiert
' RastPort bekommen wir von WINDOW() Funktion

Status$ = " Status: Auflösung wählen"+CHR$(0)
' Statuszeile aufbauen
dummy = SetWindowTitles(NWBase&,SADD(Status$),0)

DarStell$(0) =  "     Normal     "
DarStell$(1) =  " Hold and Modify"
DarStell$(2) =  " Extra Halfbrite"

XAuff$(0) = "   Normal"
XAuff$(1) = "   HIRES "

YAuff$(0) = "   Normal  "
YAuff$(1) = " Interlaced"

x(0) = 0:x(1) = 0:x(2) = 0:x(3) = 4
' Welche Werte, bzw. Strings an welcher Position?

y = 0
' oberste Zeile = Startzeile

Modulo(0) = 3
Modulo(1) = 2
Modulo(2) = 2
Modulo(3) = 6
' Wieviele Werte fuer jede Auswahlzeile?

COLOR 1

Farbe (0) = 1
Farbe (1) = 2
Farbe (2) = 2
Farbe (3) = 2
' Zeile Eins => weiß restliche Zeilen => blau
```

```
Ausg:
  LOCATE 10,17
  COLOR Farbe(0)
  PRINT  "Darstellungsmodus : ";DarStell$(x(0))
  LOCATE 12,21
  COLOR Farbe(1)
  PRINT  "X-Auflösung : ";XAuff$(x(1))
  LOCATE 14,21
  COLOR Farbe(2)
  PRINT "Y-Auflösung : ";YAuff$(x(2))
  LOCATE 16,24
  COLOR Farbe(3)
  PRINT  "BitPlanes : ";x(3)+1

  Eingabe: a$ = INKEY$
  IF a$ = "" THEN Eingabe

  IF a$ = CHR$(30) THEN x(y) = (x(y)+1) MOD Modulo(y)
  IF a$ = CHR$(31) THEN x(y) = x(y)-1
  ' CRSR Links/Rechts

  IF x(y) < 0 THEN x(y) = Modulo(y)-1
  IF a$ = CHR$(28) THEN
    Farbe(y) = 2
    y = y-1
  END IF
  ' CRSR Oben

  IF a$ = CHR$(29) THEN
    Farbe(y) = 2
    y = (y+1) MOD 4
  END IF
  ' CRSR Unten

  IF y<0 THEN y=3

  Farbe(y) = 1

  IF (x(0) > 0) AND (y=0) THEN
    x(1) = 0                ' HAM oder Halfbrite ausgewählt
    x(3) = 5                ' 5+1 BitPlanes
    Modulo(3) = 6
  END IF
```

```
IF x(0)>0 THEN x(3) = 5 ' Anzahl BitPlanes wird bei HAM oder
                        ' Halfbrite nicht verändert

IF (x(0) = 0) AND (x(1)<>1) THEN
   Modulo(3) = 5
   IF x(3) > 4 THEN x(3) = 4  ' Maximal 4+1 BitPlanes
END IF                        ' im Normal Modus

IF (x(1) = 1) AND (y = 1) THEN
  x(0) = 0                    ' HIRES und Normal
  IF x(3)>3 THEN x(3) = 3     ' Maximal 3+1 BitPlanes
  Modulo(3) = 4
END IF

IF a$<> CHR$(13) THEN GOTO Ausg

COLOR 1

RasterW%=(x(1)+1)*320
RasterH%=(x(2)+1)*200 'PAL-Auflösung

IF x(0) = 1 THEN Modus% = &H800      'HAM
IF x(0) = 2 THEN Modus% = &H80       'HalfBrite
IF x(1) = 1 THEN Modus% = &H8000     'HIRES
' (HAM, Halfbrite) und HIRES schließen sich gegenseitig aus
IF x(2) = 1 THEN Modus% = Modus% OR 4 'LACE

CLS
RasterT% = x(3)+1

IF RasterT% = 6 THEN RasterT% = 5
' Fals HAM, erst einmal einen normalen Screen
' (max. 5 BitPlanes) eröffnen

IF FRE(-1) < (RasterW%/8*RasterH%*RasterT%)+5000 THEN
  CLS
  CALL Ausgabe ("Speicher reicht für die BitMap nicht aus!!!",
  100,False)
  CALL Ausgabe ("Bitte Auflösung und/oder Anzahl der BitMaps
  vermindern!!!",130,False)

CALL DialogBox("Ok",1,True,x1b%,y1b%,x2b%,y2b%,False)
CALL DoDialog(n%,1,-1,-1,-1,-1,x1b%,y1b%,x2b%,y2b%,-1,-1,-1,-1)
  CLS
  GOTO Ausg
END IF
```

```
SCREEN 1,RasterW%,RasterH%,RasterT%,x(1)+(x(2)*2)+1
  '     Breite  ,Höhe    ,Tiefe  ,Modus

WINDOW 2,"",(0,0)-(RasterW%-9,RasterH%-15),0,1
  ' Window ueberlagern

RasterT% = x(3)+1
  ' alter Wert für RasterT

WBase& = WINDOW(7)
WScreen& = PEEKL(WBase&+46)
  ' Adresse des neuen Screens aus Window-Adresse extrahieren

POKEL WBase&+24,&H100 OR &H800 OR 65536&
  ' Window Flags verändern: Borderless, RBMTrap, NoCareRefresh

Viewport& = PEEKL(WBase&+46)+44
ColorMap& = PEEKL(Viewport&+4)
RastPort& = Viewport&+40
BitMap& = WScreen&+184

Adr = PEEKL(RastPort&+4)+8 ' BitPlanes fuer BildLaden  FOR i=0 TO
RasterT%-1                 ' und Speichern
  BitPlanes&(i) = PEEKL(Adr+4*i)
NEXT i

BitPlane6& = BitMap&+28
' Wohin Adresse der sechsten BitPlane schreiben?

Depth& = BitMap&+5
' Wohin mit der evt. neuen Tiefe?

CALL SetRast&(RastPort&,0)
CALL SetRGB4&(Viewport&,1,15,15,15)
CALL SetRGB4&(Viewport&,0,0,0,0)
' Neuen Screen löschen und Hintergrundfarbe setzen

IF RasterT% = 6 THEN
  Groesse& = RasterW%*(RasterH%\8)
  Flags& = 65536&+2                  ' MEMF_CHIP | MEMF_CLEAR
  Mem& = AllocMem&(Groesse&,Flags&)
  IF Mem& = 0 THEN
    CALL Scroff
    CALL Ausgabe("Nicht genug Speicher für sechste BitPlane!!!",
    100,False)
```

```
    CALL DialogBox("Schade!",1,True,x1a%,y1a%,x2a%,y2a%,False)

    CALL DoDialog(n%,1,-1,-1,-1,-1,x1a%,y1a%,x2a%,y2a%,-1,-1,-1,-1)
    GOSUB CloseIt
    RUN
  END IF

  POKE Depth&,6
    ' Neue Tiefe

  POKEL BitPlane6&,Mem&
  BitPlanes&(5)=Mem&
    ' Adresse der sechsten BitPlane POKEN

  POKEW Viewport&+32,Modus%
    ' Darstellungsmodus POKEN

  dummy = RemakeDisplay(dummy&)
' Neues Display berechnen
END IF

Status$ = " Status: Initialisieren"+CHR$(0)
CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

CALL Scroff
CALL SetDrMd&(RastPort&,1)

RasterW1% = RasterW%-1  ' 319/639
RasterH1% = RasterH%-1  ' 199/399
RasterW2% = RasterW%/2  ' Bildmitte
RasterH2% = RasterH%/2

FaktorX = (x(1)+1)/2
FaktorY = (x(2)+1)/2

POKEW OSModus&,Modus%
POKEL OSRastPort&,RastPort&
POKEW OSRasterW1&,RasterW1%
POKEW OSRasterH1&,RasterH1%

MaxFarben% = 2^RasterT%
IF (RasterT% = 6) THEN
  IF (Modus% AND &H800) = &H800 THEN
    MaxFarben% = 16                     'HAM
  ELSE
```

```
    MaxFarben% = 64                       'Halfbrite
  END IF
END IF

POKEW OSMaxFarben&,2^RasterT%  'falls HAM: 64 Farben!
DIM Farben(MaxFarben%,3)

FOR m%=0 TO 3
  Farben(1,m%)=0         'Schwarz
  Farben(2,m%)=1         'Weiß
NEXT m%

IF (Modus% AND &H80) = &H80 THEN  'HalfBrite
  MF%=MaxFarben%/2

  Farben(MF%+1,0)=MF%
  Farben(MF%+2,0)=MF%+1
  FOR m%=1 TO 3
    Farben(MF%+1,m%)=0
    Farben(MF%+2,m%)=.5
  NEXT m%
  RESTORE OptimaleHBFarben
ELSE
  MF%=MaxFarben%
  RESTORE OptimaleFarben
END IF

FOR n% = 3 TO MF%
  READ r%,g%,b%

  Farben(n%,0)=n%-1
  Farben(n%,1)=r%/15
  Farben(n%,2)=g%/15
  Farben(n%,3)=b%/15
  CALL SetRGB4&(Viewport&,n%-1,r%,g%,b%)

  IF (Modus% AND &H80)<>0 THEN          'HalfBrite
    Farben(n%+MF%,0)=n%+MF%
    FOR m%=1 TO 3
      Farben(n%+MF%,m%)=(INT(Farben(n%,m%)*15)\2)/15
    NEXT m%
  END IF

NEXT n%
' => Farben(n,0)     = Farbindex,
'    Farben(n,1..3) = RGB Helligkeitsstufen
```

```
 ' Nun noch die Farben in das Feld fuer die Assemblerroutine POKEN:

  FOR n%=0 TO MaxFarben%-1
    POKEW OSFarben&+n%*8,Farben(n%+1,0)
    FOR m%=1 TO 3
      POKEW OSFarben&+n%*8+m%*2,Farben(n%+1,m%)*1024
    NEXT m%
  NEXT n%
  CLS
RETURN
```

Weiterhin wird in 'RasterInit' dafür gesorgt, daß die Farben für den Display-Screen korrekt gesetzt werden. Sollten Ihnen die von uns eingestellten Default-Farbwerte nicht zusagen, können Sie diese nachträglich ändern:

```
FarbPalette:
  Status$ = " Status: Farben ändern"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  CALL SetRast&(RastPort&,0)
  CALL Scron
  CALL Ausgabe("Welche Farbe ändern ?",100,True)

  AnzCol = MaxFarben%
  IF AnzCol = 64 THEN AnzCol = 32 ' Halfbrite ?
  Breite = (RasterW%-40)/AnzCol
  LINE ((20+1),1)-(20+Breite-1,RasterH%/10-1),1,b
  FOR i=1 TO AnzCol-1
     LINE ((20+i*Breite+1),1)-((20+(i+1)*Breite)-1,RasterH%/10-1),i,BF
      IF MaxFarben% = 64 THEN  ' Halfbrite Farben ausgeben
        LINE ((20+i*Breite+1),RasterH%/10)-((20+(i+1)*Breite)-
        1,RasterH%/5-1),i+32,BF
      END IF
  NEXT

 LoopA:
  CALL EmptyBuffers
```

```
WHILE MOUSE(0) = 0
WEND

x = MOUSE (1)
y = MOUSE (2)

IF (x < 20) OR (x > Rasterw%-20) THEN
  GOTO LoopA:
IF (y < 1) OR (y > Rasterh%/10) THEN
  GOTO LoopA:
END IF

Farb% = INT ((x-20)/Breite)  ' welche Farbe angeklickt
CALL Scroff
rot=INT(Farben(Farb%+1,1)*15.5)
gruen=INT(Farben(Farb%+1,2)*15.5)
blau=INT(Farben(Farb%+1,3)*15.5)

LOCATE 10,1
PRINT " Rot -Komponente (0..15): ";
CALL FormInputInt (rot,30!,0!,15!)

LOCATE 11,1
PRINT " Grün-Komponente (0..15):";
CALL FormInputInt (gruen,30!,0!,15!)

LOCATE 12,1
PRINT " Blau-Komponente (0..15):";
CALL FormInputInt (blau,30!,0!,15!)

Farben(Farb%+1,1)=rot/15
Farben(Farb%+1,2)=gruen/15
Farben(Farb%+1,3)=blau/15

POKEW OSFarben&+Farb%*8+2,rot/15*1024
POKEW OSFarben&+Farb%*8+4,gruen/15*1024
POKEW OSFarben&+Farb%*8+6,blau/15*1024

IF MaxFarben%=64 THEN                  'ExtraHalfBrite
  Farben(Farb%+33,1)=(rot\2)/15
  Farben(Farb%+33,2)=(gruen\2)/15
  Farben(Farb%+33,3)=(blau\2)/15

POKEW OSFarben&+Farb%*8+258,(rot\2)/15*1024
POKEW OSFarben&+Farb%*8+260,(gruen\2)/15*1024
POKEW OSFarben&+Farb%*8+262,(blau\2)/15*1024
```

```
  END IF

  CALL SetRGB4&(Viewport&,Farb%,CINT(rot),CINT(gruen),CINT(blau))

  CALL DialogBox ("Ende",0,True,x1a%,y1a%,x2a%,y2a%,False)
  CALL DialogBox ("Nächste Farbe",2,False,x1b%,y1b%,x2b%,y2b%,False)

  CALL DoDialog (n%,0,x1a%,y1a%,x2a%,y2a%,-1,-1,-1,-1,x1b%,y1b%,
  x2b%,y2b%)
  CLS

  IF n% = 2 THEN
    CALL Scron
    GOTO LoopA
  END IF
  Neu! = True
  Hg = False
  GOSUB MakeMenu
Return
```

Die Farben des Benutzer-Screens können Sie, wenn Sie wollen, in der Routine 'Rasterinit' mittels 'PALETTE'-Anweisungen ändern.

Für den Datenaustausch zwischen Computer und Floppy haben wir natürlich auch gesorgt. Sie können nämlich z.B. vorher mit dem Editor erstellte oder die schon vorhandenen Objektdefinitionen (im 'Objekte' Unterverzeichnis der beigelegten Diskette) von Diskette laden. Oder Sie verbinden (mergen) verschiedene Objektdefintionen und speichern alle Objekte dann zusammen auf Diskette ab.

```
Laden:
  ' Array K() von Diskette laden
  GOSUB Directory

  Status$ = " Status: Objekt laden"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
```

```
  LOCATE 3,1
  PRINT "  Welche Objekte wollen Sie laden?"
  PRINT
  PRINT "      Filename :";
  dn$ = ""
  CALL FormInputString (dn$,30!)

  CLS
  IF dn$ <> "" THEN
    dn$ = dn$+".LIST"
    OPEN "I",#1,dn$,1024
    INPUT #1,AnzahlK
    LOCATE 10,10
    PRINT "Insgesamt ";AnzahlK;" Objekte ("+dn$+")"
    PRINT
    PRINT "       Maximale Anzahl der Objekte?";
    MaxAnzahl = AnzahlK
    CALL FormInputInt (MaxAnzahl,30!,1!,AnzahlK)

    IF AnzahlK > MaxAnzahl THEN AnzahlK = MaxAnzahl

    ERASE K
    DIM K(MaxAnzahl,5,2)

    FOR n%=1 TO AnzahlK
      FOR p%=0 TO 5
        INPUT #1,K(n%,p%,0),K(n%,p%,1),K(n%,p%,2)
      NEXT p%
    NEXT n%
    CLOSE #1
    Neu! = True
    Start% = 1
  END IF
  CLS
  GOSUB MakeMenu
RETURN

Speichern:
  ' Speichere Array K() auf Diskette
  GOSUB Directory

  Status$ = " Status: Objekt speichern"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
```

```
  LOCATE 3,1
  PRINT "  Unter welchem Namen wollen Sie die Objekte abspeichern?"
  PRINT
  PRINT "      Filename :";
  dn$ = ""
  CALL FormInputString (dn$,30!)

  CLS
  IF dn$<>""THEN
    dn$ = dn$ + ".LIST"
    OPEN "O",#1,dn$,1024
    PRINT #1,AnzahlK
    FOR n%=1 TO AnzahlK
      FOR p%=0 TO 5
        PRINT #1,K(n%,p%,0),K(n%,p%,1),K(n%,p%,2)
      NEXT p%
    NEXT n%
    CLOSE #1
'Wem nicht gefällt, daß beim Abspeichern Icons erzeugt werden:
'    dn$ = dn$+".info"
'    KILL dn$
  END IF
  CLS
  GOSUB MakeMenu
RETURN

Mergen:
'   Elemente dem vorhandenen Array K() hinzufuegen
  IF (MaxAnzahl-AnzahlK) <= 0 THEN
    Status$ = " Status: Objekt mergen"+CHR$(0)
    CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

    a$ = "Kein Platz mehr für weitere Objekte!!!"
    CALL Ausgabe (a$,100,False)
    a$ = "Bitte erst mit 'New' ein genügend großes Array K() anlegen!"
    CALL Ausgabe (a$,130,False)
    CALL DialogBox("OK",1,True,x1a%,y1a%,x2a%,y2a%,False)

    CALL DoDialog(n%,1,-1,-1,-1,-1,x1a%,y1a%,x2a%,y2a%,-1,-1,-1,-1)
  ELSE
    GOSUB Directory

    Status$ = " Status: Objekt mergen"+CHR$(0)
    CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
```

```
    LOCATE 3,1
    PRINT " Welche Objekte wollen Sie mergen?"
    PRINT
    dn$ = ""
    PRINT "       Filename :";
    CALL FormInputString (dn$,30!)

    CLS
    IF dn$ <> "" THEN
      dn$ = dn$ + ".LIST"
      OPEN "I",#1,dn$,1024
      INPUT #1,x

      LOCATE 6,1
      PRINT "         Insgesamt ";x;" Objekte ("+dn$+")."
      PRINT
      PRINT "         Noch Platz f³r ";MaxAnzahl-AnzahlK;" Objekte."
      PRINT
      PRINT "    Wieviele mergen? ";
      anz = x
      CALL FormInputInt (anz,30!,1!,1000!)

      IF x<anz THEN anz = x
      IF MaxAnzahl-AnzahlK<anz THEN anz = MaxAnzahl-AnzahlK

      IF anz>=1 THEN
        FOR n%=AnzahlK+1 TO AnzahlK+anz
          FOR p%=0 TO 5
            INPUT #1,K(n%,p%,0),K(n%,p%,1),K(n%,p%,2)
          NEXT p%
        NEXT n%
        AnzahlK = AnzahlK+anz
        Neu! = True
        Start% = 1
      END IF
      CLOSE #1
    END IF
  END IF
  CLS
  GOSUB MakeMenu
RETURN
```

Beim 'Mergen' von Objekten sollten Sie allerdings darauf achten, daß das Array K(), in dem ja bekanntlich die Objekte intern

abgespeichert werden, groß genug ist. Die Anzahl der Array-Elemente wird in 'FeldInit' festgesetzt. Die eventuell schon vorhandenen Objektdefinitionen gehen dabei aber verloren!

```
FeldInit:
  ' Neues Array K() anlegen
  Status$ = " Status: Neues Feld anlegen"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  LOCATE 10,1
  PRINT "         Maximale Anzahl der Objekte =";
  a = MaxAnzahl
  CALL FormInputInt (a,30!,0!,1000!)

  IF MaxAnzahl <> a THEN
    MaxAnzahl = a
    AnzahlK=0
    ERASE K
    DIM K(MaxAnzahl,5,2)
    Start% = 0
  END IF

  CLS
  GOSUB MakeMenu
RETURN
```

Neben den Objektdefinitionen können Sie allerdings auch neue Materialkonstanten von Diskette laden:

```
LoadMat:
  GOSUB Directory

  Status$ = " Status: Materialkonstanten laden"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  LOCATE 3,1
  PRINT "  Welche Materialien wollen Sie laden?"
  PRINT
  PRINT "      Filename :";
```

```
    dn$ = ""
    CALL FormInputString (dn$,30!)

    IF dn$<>"" THEN
      dn$=dn$+".MAT"
      OPEN "i",#1,dn$
      matptr = 1
      INPUT #1,AnzahlMat
      ERASE Mat
      DIM Mat(AnzahlMat,6)
      WHILE NOT(EOF(1))
        FOR i=0 TO 6
          INPUT #1,Mat(matptr,i)
        NEXT i
        matptr = matptr+1
      WEND
      CLOSE #1
    END IF
    CLS
    GOSUB MakeMenu
  RETURN
```

Für etwas länger andauernde Freude an den erzeugten Bildern haben wir eine Routine für Sie eingerichtet, die es ermöglicht, das Tracer-Erzeugnis im IFF-Format (Interchange-File-Format) abzuspeichern. Das so abgespeicherte Bild können Sie dann mit jedem Malprogramm, das auch das IFF-Format unterstützt (z.B. mit DeluxePaint), weiter bearbeiten:

```
BildSpeichern:
  GOSUB Directory

  Status$ = " Status: Bild abspeichern"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  LOCATE 3,1
  PRINT "  Unter welchem Namen wollen Sie das Bild abspeichern?"
  PRINT
  PRINT "      Filename:";
  IFFFile$ =""
  CALL FormInputString (IFFFile$,30!)
```

```
IF IFFFile$ <> "" THEN

  Handle& = 0      ' File Handle
  Buffer& = 0      ' Bufferspeicher
  ' Buffer reservieren
  Flags& = 65537&    ' MEMF_PUBLIC | MEMF_CLEAR
  BufferSize& = 360
  Buffer& = AllocMem&(BufferSize&,Flags&)
  IF Buffer& = 0 THEN
    Dialog$ = "Kein Speicher mehr!!!"
    GOTO EndSave
  END IF

  ColorBuffer& = Buffer&

  Null& = 0
  PadByte%  = 0

  IF RasterW% = 320 THEN
    IF RasterH% = 200 THEN
      Aspect% = &HA0B
    ELSE
      Aspect% = &H140B
    END IF
  ELSE
    IF RasterH% = 200 THEN
      Aspect% = &H50B
    ELSE
      Aspect% = &HA0B
    END IF
  END IF

  IFFFile$ = IFFFile$ + CHR$(0)
  Handle& = xOpen&(SADD(IFFFile$),1006)
  IF Handle& = 0 THEN
    Dialog$ = "Kann angegebenes File nicht eröffnen!!!"
    GOTO EndSave
  END IF


  ' Wieviel Bytes enthalten die einzelnen IFF-Chunks?
  BMHDSize& = 20
  CMAPSize& = MaxFarben%*3 + ((MaxFarben%*3) AND 1)
  CAMGSize& = 4
  BODYSize& = (RasterW%/8)*RasterH%*RasterT%
  ' FORMsize& = Chunk-Längen + 8 Bytes pro Chunk-Header + ' 4 Bytes
```

```
("ILBM")
FORMSize& = BMHDSize&+CMAPSize&+CAMGSize&+BODYSize&+36

' FORM-Header
Chunk$ = "FORM"
Length& = xWrite&(Handle&,SADD(Chunk$),4)
Length& = xWrite&(Handle&,VARPTR(FORMSize&),4)
' + ILBM fuer BitMap-File
Chunk$ = "ILBM"
Length& = xWrite&(Handle&,SADD(Chunk$),4)

IF Length& <= 0 THEN
  Dialog$ = "Schreibfehler bei FORM-Header!!!"
  GOTO EndSave
END IF

' BMHD-Chunk
Chunk$ = "BMHD"
Length& = xWrite&(Handle&,SADD(Chunk$),4)
Length& = xWrite&(Handle&,VARPTR(BMHDSize&),4)
Length& = xWrite&(Handle&,VARPTR(RasterW%),2)
Length& = xWrite&(Handle&,VARPTR(RasterH%),2)
Length& = xWrite&(Handle&,VARPTR(Null&),4)
Temp% = (256 * RasterT%) ' Kein MASKING
Length& = xWrite&(Handle&,VARPTR(Temp%),2)
Temp% = 0                 ' kein Packing
Length& = xWrite&(Handle&,VARPTR(Temp%),2)
Temp% = 0
Length& = xWrite&(Handle&,VARPTR(Temp%),2)
Length& = xWrite&(Handle&,VARPTR(Aspect%),2)
Length& = xWrite&(Handle&,VARPTR(RasterW%),2)
Length& = xWrite&(Handle&,VARPTR(RasterH%),2)

IF Length& <= 0 THEN
  Dialog$ = "Schreibfehler bei BMHD-Chunk!!!"
  GOTO EndSave
END IF

' CMAP-Chunk
Chunk$ = "CMAP"
Length& = xWrite&(Handle&,SADD(Chunk$),4)
Length& = xWrite&(Handle&,VARPTR(CMAPSize&),4)

FOR i%=0 TO MaxFarben%-1
  Farbe% = GetRGB4%(ColorMap&,i%)
  blau = Farbe% AND 15
```

```
    Farbe% = Farbe% - blau
    gruen = (Farbe%/16) AND 15
    Farbe% = Farbe%-gruen*16
    rot = (Farbe%/256) AND 15
    POKE(ColorBuffer&+(i%*3)),rot*16
    POKE(ColorBuffer&+(i%*3)+1),gruen*16
    POKE(ColorBuffer&+(i%*3)+2),blau*16
  NEXT

  Length& = xWrite&(Handle&,ColorBuffer&,CMAPSize&)

  IF Length& <= 0 THEN
    Dialog$ = "Schreibfehler bei CMAP-Chunk!!!"
    GOTO EndSave
  END IF

  ' CAMG-Chunk
  Chunk$ = "CAMG"
  Length& = xWrite&(Handle&,SADD(Chunk$),4)
  Length& = xWrite&(Handle&,VARPTR(CAMGSize&),4)
  Modes& = PEEKW(Viewport& + 32)
  Length& = xWrite&(Handle&,VARPTR(Modes&),4)

  IF Length& <= 0 THEN
    Dialog$ = "Schreibfehler bei CAMG-Chunk!!!"
    GOTO EndSave
  END IF

  ' BODY-Chunk (BitMaps)
  Chunk$ = "BODY"
  Length& = xWrite&(Handle&,SADD(Chunk$),4)
  Length& = xWrite&(Handle&,VARPTR(BODYSize&),4)
  BytesPerRow% = RasterW%/8
  FOR y1 = 0 TO RasterH%-1
    FOR b=0 TO RasterT%-1
      Adress& = BitPlanes&(b)+(y1*BytesPerRow%)
      Length& = xWrite&(Handle&,Adress&,BytesPerRow%)
      IF Length& <= 0 THEN
        Dialog$ = "Schreibfehler bei BODY-Chunk!!!"
        GOTO EndSave
      END IF
    NEXT
  NEXT

  Dialog$ = "Savings OK"
```

```
  EndSave:
    IF Handle& <> 0 THEN CALL xClose&(Handle&)
    IF Buffer& <> 0 THEN CALL FreeMem&(Buffer&,BufferSize&)
    CALL DialogBox(Dialog$,1,True,x1b%,y1b%,x2b%,y2b%,False)
    CALL DoDialog(n%,1,-1,-1,-1,-1,x1b%,y1b%,x2b%,y2b%,-1,-1,-1,-1)
  END IF
  CLS
  GOSUB MakeMenu
RETURN
```

Die von dieser Routine erzeugten Files sind keinem Komprimierungsverfahren unterzogen worden. So wird zwar mehr Speicherplatz auf der Diskette benötigt, aber das Abspeichern geht wesentlich schneller vonstatten (ca. 75% Geschwindigkeitssteigerung).

Auch an die Ausgabe des Bildes auf einen Drucker haben wir gedacht. Die folgende Hardcopy-Routine gibt das Bild auf den Drucker aus, der vorher mit 'Preferences' ausgewählt (hoffentlich an Ihren Amiga angeschlossen) worden ist:

```
HardCopy:
  Status$ = " Status: Bild ausdrucken"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  CALL DialogBox("Kein Printer",0,True,x1a%,y1a%,x2a%,y2a%,False)
  CALL DialogBox("Printer OK",2,False,x1c%,y1c%,x2c%,y2c%,False)
  CALL DoDialog(n%,0,x1a%,y1a%,x2a%,ya2%,-1,-1,-1,-1,x1c%,y1c%,x2c%,
  y2c%)
  CLS
  IF n% <> 0 THEN
    Modes% = PEEKW(Viewport& + 32)

    sigBit% =  AllocSignal%(-1)
    Flags& = 65537&              ' MEMF_PUBLIC | MEMF_CLEAR
    MsgPort& = AllocMem&(40,Flags&)
    IF MsgPort& = 0 THEN
      Dialog$ = "Kein 'MsgPort'!!!"
      GOTO EndDrucker4
    END IF
```

```
   POKE(MsgPort& + 8), 4
   POKE(MsgPort& + 9), 0
   Nam$ = "PrtPort"+CHR$(0)
   POKEL(MsgPort& + 10), SADD(Nam$)
   POKE(MsgPort& + 14), 0
   POKE(MsgPort& + 15), sigBit%
   SigTask& = FindTask&(0)
   POKEL(MsgPort& + 16), SigTask&

   CALL AddPort(MsgPort&)  'Port anmelden

   ioRequest& = AllocMem&(64,Flags&)
   IF ioRequest& = 0  THEN
     Dialog$ = "Kein ioRequest!!!"
     GOTO EndDrucker3
   END IF

   POKE(ioRequest& + 8),5
   POKE(ioRequest& + 9),0
   POKEL(ioRequest& + 14), MsgPort&


   Nam$ = "printer.device"+CHR$(0)
   DruckerError& = OpenDevice&(SADD(Nam$),0,ioRequest&,0)
   IF DruckerError& <> 0  THEN
     Dialog$ = "Kein Drucker?!?"
     GOTO EndDrucker2
   END IF

   POKEW(ioRequest& + 28), 11          ' DumpRastport an Drucker senden
POKEL(ioRequest& + 32), RastPort&      ' Ganzen Screen drucken
POKEL(ioRequest& + 36), ColorMap&
   POKEL(ioRequest& + 40), Modes%
   POKEW(ioRequest& + 44), 0
   POKEW(ioRequest& + 46), 0
   POKEW(ioRequest& + 48), RasterW%
   POKEW(ioRequest& + 50), RasterH%
   POKEL(ioRequest& + 52), 0&
   POKEL(ioRequest& + 56), 0&
   POKEW(ioRequest& + 60), &H84

   CALL DialogBox("Printing...",1,False,x1a%,y1a%,x2a%,y2a%,False)
```

```
    DruckerError& = DoIO&(ioRequest&)
    IF DruckerError& <> 0 THEN
 Dialog$ = "DumpRPort Fehler ="+STR$(DruckerError&)+"!!!"        GOTO
 EndDrucker
    END IF

    CLS
    Dialog$ = "Hardcopy done"

    EndDrucker:
      CALL CloseDevice(ioRequest&)

    EndDrucker2:
      POKE(ioRequest& + 8), &HFF
      POKEL(ioRequest& + 20), -1
      POKEL(ioRequest& + 24), -1
      CALL FreeMem&(ioRequest&,64)

    EndDrucker3:
      CALL RemPort(MsgPort&)
      POKE(MsgPort& + 8), &HFF
      POKEL(MsgPort& + 20), -1
      CALL FreeSignal(sigBit%)
      CALL FreeMem&(MsgPort&,40)

    EndDrucker4:
      CALL DialogBox(Dialog$,1,True,x1b%,y1b%,x2b%,y2b%,False)
      CALL DoDialog(n%,1,-1,-1,-1,-1,x1b%,y1b%,x2b%,y2b%,-1,-1,-1,-1)
  END IF
  CLS
  GOSUB MakeMenu
RETURN
```

Und weil wir uns gedacht haben, daß ein Hintergrund für die zu schattierenden Objekte eine feine Sache sei, haben wir mit den Routinen 'Hintergrund', 'Himmel', 'Fhg' (fließender Hintergrund) und 'BildLaden' die Möglichkeit geschaffen, einen neuen Hintergrund anstatt der sonst üblichen Hintergrundfarbe zu erzeugen.

Dabei können Sie wählen zwischen einem einfachen Muster, einem 'Sternenhimmel', einem fließenden Hintergrund, der aus

Farb- bzw. Helligkeitsabstufungen besteht, und aus einem IFF-Bild, das z.B. vorher mit einem Zeichenprogramm erstellt wurde.

```
Hintergrund:
  ' Hintergrund bestimmen
  Status$ = " Status: Hintergrund auswählen"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  CALL DialogBox("Muster",0,True,x1a%,y1a%,x2a%,y2a%,False)
  CALL DialogBox("Bild Laden",2,False,x1b%,y1b%,x2b%,y2b%,False)

  CALL DoDialog(n%,0,x1a%,y1a%,x2a%,y2a%,-1,-1,-1,-1,x1b%,y1b%,
  x2b%,y2b%)
  CLS

  IF n%=2 THEN
    GOSUB BildLaden
    Hg = True
  ELSE
    CALL DialogBox("Muster",0,True,x1a%,y1a%,x2a%,y2a%,False)
    CALL DialogBox("Himmel",1,False,x1b%,y1b%,x2b%,y2b%,False)
    CALL DialogBox("Fließend",2,False,x1c%,y1c%,x2c%,y2c%,False)

    CALL DoDialog(n%,0,x1a%,y1a%,x2a%,y2a%,x1b%,y1b%,x2b%,y2b%,x1c%,
    y1c%,x2c%,y2c%)
    CLS

    IF n% = 0 THEN

      CALL Ausgabe("Hintergrund = Muster",40,False)

      LOCATE 10,1
      PRINT "      Nummer des Musters (0..34):";
      a = 0
      CALL FormInputInt (Hx,30!,0!,34!)
      Muster% = a

      LOCATE 11,1
      PRINT "      Vordergrundstift (APen):";
      a=APen%
      CALL FormInputInt (a,30!,0!,CSNG(MaxFarben%-1))
      APen% = a
```

```
      LOCATE 12,1
      PRINT "      Hintergrundstift (BPen):";
      a = BPen%
      CALL FormInputInt (a,30!,0!,CSNG(MaxFarben%-1))
      BPen% = a

 Muster% = Muster% MOD (AnzahlMusterS%+AnzahlMusterX%*2+1)
      IF Muster% > AnzahlMusterS% THEN
        CALL ExtendedFill(0,0,RasterW1%,RasterH1%,MusterHX(Muster%-
        AnzahlMusterS%),0!,1!,APen%,BPen%)
       ELSE
        CALL
StandardFill(0,0,RasterW1%,RasterH1%,MusterHS(AnzahlMusterS%-
Muster%),APen%,BPen%)
      END IF
      CLS
      Hg = True
    END IF

    IF n% = 1 THEN
      LOCATE 3,1
      PRINT " Himmel Art:"
      PRINT "  0 = Sterne   (verstreut)"
      PRINT "  1 = Sterne   (zentriert)"
      PRINT "  2 = Strahlen (verstreut)"
      PRINT "  3 = Strahlen (zentriert)"
      PRINT "  4 = Strahlen (mittenzentriert)"

      LOCATE 10,1
      PRINT " Art    :";
      a = Art%
      CALL FormInputInt (a,30!,0!,4!)
      Art% = a

      LOCATE 11,1
      PRINT " Farbe  :";
      a = 1
      CALL FormInputInt (a,30!,0!,CSNG(MaxFarben%-1))
      HFarb% = a

      LOCATE 12,1
      PRINT " Anzahl :";
      a = 100
      CALL FormInputInt (a,30!,0!,500!)
      anz% = a
```

```
      CLS
      CALL Himmel (Art%,HFarb%,anz%)
      Hg = True
    END IF

    IF n% = 2 THEN
      CLS

      CALL Ausgabe("Fließender Hintergrund",40,False)

      LOCATE 10,1
      PRINT "         von Farbe:";
      F1 = 0
      CALL FormInput (F1,30!,0!,CSNG(MaxFarben%-1))
      Farbe1%=F1
      LOCATE 12,1
      PRINT "         bis Farbe:";
      F2 = 1
      CALL FormInput (F2,30!,0!,CSNG(MaxFarben%-1))
      Farbe2%=F2

      CALL Fhg(Farbe1%,Farbe2%)
      Hg = True
    END IF
  END IF
  CLS
  GOSUB MakeMenu
RETURN


SUB Fhg(Farbe1%,Farbe2%) STATIC
SHARED RasterW1%,RasterH1%,Farben(),MaxFarben%,WScreen&
SHARED NWBase&,RastPort&,OSSetPoint&,OSModus&,OSMaxFarben&
 ' Produziert einen fließenden Übergang von Farbe1 nach Farbe2
   CALL Scron

  Rot.s=Farben(Farbe1%+1,1)
  Gruen.s=Farben(Farbe1%+1,2)
  Blau.s=Farben(Farbe1%+1,3)
  Rot.e=Farben(Farbe2%+1,1)
  Gruen.e=Farben(Farbe2%+1,2)
  Blau.e=Farben(Farbe2%+1,3)

  IF (PEEKW(OSModus&) AND &H800) = &H800 THEN  'HAM
```

```
    'Hintergrund nur mit 16 Farben, da sonst von Objekten zerstört.

    POKEW OSMaxFarben&,16
    FOR y%=0 TO RasterH1%
      rot=Rot.s+(y%/RasterH1%)*(Rot.e-Rot.s)
      gruen=Gruen.s+(y%/RasterH1%)*(Gruen.e-Gruen.s)
      blau=Blau.s+(y%/RasterH1%)*(Blau.e-Blau.s)
'      CALL SetPoint(0,y%,RasterW1%,y%,rot,gruen,blau)
      CALL OSSetPoint&(0,y%,RasterW1%,y%,CLNG(1024*rot),
      CLNG(1024*gruen),CLNG(1024*blau))
    NEXT y%
    POKEW OSMaxFarben&,64
  ELSE
    FOR y%=0 TO RasterH1%
      rot=Rot.s+(y%/RasterH1%)*(Rot.e-Rot.s)
      gruen=Gruen.s+(y%/RasterH1%)*(Gruen.e-Gruen.s)
      blau=Blau.s+(y%/RasterH1%)*(Blau.e-Blau.s)
'      CALL SetPoint(0,y%,RasterW1%,y%,rot,gruen,blau)
      CALL OSSetPoint&(0,y%,RasterW1%,y%,CLNG(1024*rot),
      CLNG(1024*gruen),CLNG(1024*blau))
    NEXT y%
  END IF
  CALL Scroff
END SUB


SUB Himmel(a%,Col%,anz%) STATIC
SHARED WScreen&,NWBase&,RasterW%,RasterH%,RasterW1%
SHARED RasterH1%,RasterW2%,RasterH2%,RastPort&
  CALL Scron

  CALL SetAPen&(RastPort&,Col%)
  CALL SetRast&(RastPort&,0)

  FOR n%=1 TO anz%
    RANDOMIZE TIMER

    IF a%<4 THEN
      IF a% AND 1 THEN
        x%=RasterW2%+RasterW2%*RND*RND*SGN(RND-.5)
        y%=RasterH2%+RasterH2%*RND*RND*SGN(RND-.5)
      ELSE
        x%=RND*RasterW1%
        y%=RND*RasterH1%
      END IF
```

```
    IF a% AND 2 THEN
      IF x%=RasterW2% AND y%=RasterH2% THEN
        dummy = WritePixel(RastPort&,x%,y%)
      ELSE
        x1%=(x%-RasterW2%)*.1+x%
        y1%=(y%-RasterH2%)*.1+y%
        IF x1%>=0 AND x1%<RasterW% AND y1%>=0 AND y1%<RasterH% THEN
          CALL Move&(RastPort&,x%,y%)
          CALL Draw&(RastPort&,x1%,y1%)
        END IF
      END IF
    ELSE

      IF RND<.9 THEN
        CALL WritePixel&(RastPort&,x%,y%)
      ELSE
        CALL Move&(RastPort&,x%-1,y%)
        CALL Draw&(RastPort&,x%+1,y%)
        CALL Draw&(RastPort&,x%,y%-1)
        CALL Draw&(RastPort&,x%,y%+1)
      END IF
    END IF
  ELSE

    x%=RND*RasterW1%
    y%=RND*RasterH1%
    CALL Move&(RastPort&,RasterW2%,RasterH2%)
    CALL Draw&(RastPort&,x%,y%)
  END IF
NEXT n%

CALL SetAPen&(RastPort&,1)

CALL Scroff
END SUB


BildLaden:
  GOSUB Directory

  Status$ = " Status: Bild laden"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  LOCATE 3,1
    PRINT "  Welches IFF-File wollen Sie laden?"
  PRINT
```

```
PRINT "      Filename :";
IFFFile$ =""
CALL FormInputString (IFFFile$,30!)

IF IFFFile$ <> "" THEN

  BMHD = False
  CMAP = False
  CAMG = False
  BODY = False

  Handle& = 0
  Buffer& = 0

  ' Buffer reservieren
  Flags& = 65537&        ' MEMF_PUBLIC | MEMF_CLEAR
  BufferSize& = 360
  Buffer& = AllocMem&(BufferSize&,Flags&)
  IF Buffer& = 0 THEN
    Dialog$ = "Kein Speicher mehr!!!"
    GOTO EndLoad
  END IF

  EingabeBuffer& = Buffer&
  ColorBuffer& = Buffer& + 120


  IFFFile$ = IFFFile$ + CHR$(0)
  Handle& = xOpen&(SADD(IFFFile$),1005)
  IF Handle& = 0 THEN
    Dialog$ = "IFF-File kann nicht erröffnet werden!!!"
    GOTO EndLoad
  END IF


  Length& = xRead&(Handle&,EingabeBuffer&,12)
  Chunk$ = ""
  FOR n% = 8 TO 11
    Chunk$ = Chunk$ + CHR$(PEEK(EingabeBuffer&+n%))
  NEXT

  IF Chunk$ <> "ILBM" THEN
    Dialog$ = "Kein IFF-Format !!!"
    GOTO EndLoad
  END IF
```

```
LeseSchleife:
  Length& = xRead&(Handle&,EingabeBuffer&,8)
  ChunkLaenge& = PEEKL(EingabeBuffer& + 4)
  Chunk$ = ""
  FOR n% = 0 TO 3
    Chunk$ = Chunk$ + CHR$(PEEK(EingabeBuffer&+n%))
  NEXT

  IF Chunk$ = "BMHD" THEN        ' BitMap-Header
    BMHD = True
    Length& = xRead&(Handle&,EingabeBuffer&,ChunkLaenge&)
    RDepth%       = PEEK(EingabeBuffer& + 8)
    Compression%  = PEEK(EingabeBuffer& + 10)
    RWidth%       = PEEKW(EingabeBuffer& + 16)
    RHeight%      = PEEKW(EingabeBuffer& + 18)
    BytesPerRow%  = RWidth%/8
    RMaxColors%   = 2^(RDepth%)

    ' Paßt IFF-Bild auf Display-Screen ?
    IF (RWidth% <> RasterW%) OR (RHeight% <> RasterH%) THEN
      Dialog$ = "Formatfehler: "+STR$(RWidth%)+" x"+STR$(RHeight%)
      GOTO EndLoad
    END IF

  ELSEIF Chunk$ = "CMAP" THEN  ' Farb-Palette
    Length& = xRead&(Handle&,ColorBuffer&,ChunkLaenge&)
    CMAP = True
    ' Farb-Palette aufbauen
    FOR n% = 0 TO RMaxColors% - 1
      rot%   = PEEK(ColorBuffer&+(n%*3))/16
      gruen% = PEEK(ColorBuffer&+(n%*3)+1)/16
      blau%  = PEEK(ColorBuffer&+(n%*3)+2)/16
      dummy = SetRGB4(Viewport&,n%,rot%,gruen%,blau%)

      POKEW OSFarben&+n%*8+2,rot%/15*1024
      POKEW OSFarben&+n%*8+4,gruen%/15*1024
      POKEW OSFarben&+n%*8+6,blau%/15*1024
    NEXT

  ELSEIF Chunk$ = "BODY" THEN  'BitMap laden
    CALL Scron
    BODY = True
    IF Compression% = 0 THEN  'Keine Komprimierung
      FOR y1 = 0 TO RHeight% -1
        FOR b = 0 TO RDepth% -1
          IF b<RasterT% THEN
```

```
          Adress& = BitPlanes&(b)+(y1*BytesPerRow%)
        ELSE
          Adress& = Buffer&
        END IF
        Length& = xRead&(Handle&,Adress&,BytesPerRow%)
      NEXT
    NEXT

ELSEIF Compression% = 1 THEN  'CmpByteRun1 compression
  FOR y1 = 0 TO RHeight% -1
    FOR b = 0 TO RDepth% -1
      IF b<RasterT% THEN
        Adress& = BitPlanes&(b)+(y1*BytesPerRow%)
      ELSE
        Adress& = Buffer&
      END IF
      AnzBytes% = 0

      WHILE (AnzBytes% < BytesPerRow%)
        Length& = xRead&(Handle&,EingabeBuffer&,1)
        Code% = PEEK(EingabeBuffer&)
        IF Code% < 128 THEN          ' Code%-Bytes uebernehmen
          Length& = xRead&(Handle&,Adress& + AnzBytes%, Code%+1)
          AnzBytes% = AnzBytes% + Code% + 1
        ELSEIF Code% > 128 THEN      ' Byte replizieren
          Length& = xRead&(Handle&,EingabeBuffer&,1)
          Byte% = PEEK(EingabeBuffer&)
          FOR n% = AnzBytes% TO AnzBytes% + 257 - Code%
            POKE(Adress&+n%),Byte%
          NEXT
          AnzBytes% = AnzBytes% + 257 - Code%
        END IF
      WEND
    NEXT
  NEXT

  ELSE
    Dialog$ = "Unbekanntes Komprimierungsverfahren!!!"
    GOTO EndLoad
  END IF
CALL Scroff
ELSE
'   "Unbekannter" Chunk-Typ
  FOR n = 1 TO ChunkLaenge&
    Length& = xRead&(Handle&,EingabeBuffer&,1)
  NEXT
```

```
      '  Chunks haben immer gerade AnzBytes
        IF (ChunkLaenge& AND 1) = 1 THEN
          Length& = xRead&(Handle&,EingabeBuffer&,1)
        END IF
      END IF

      ' Alle Chunks gelesen?
      IF (BMHD = True) AND (CMAP = True) AND (BODY = True) THEN
        GOTO LoadOK
      END IF

      ' Lesen ok, nächsten Chunk lesen
      IF Length& > 0 THEN GOTO LeseSchleife

      IF Length& < 0 THEN
        Dialog$ = "Lesefehler!!!"
        GOTO EndLoad
      END IF

      IF (BMHD=False) OR (CMAP=False) OR (BODY=0) THEN
        Dialog$ = "Nicht alle ben tigten ILBM-Chunks gefunden!!!"
        GOTO EndLoad
      END IF

      LoadOK:
        Dialog$ = "Loading OK"

      EndLoad:
        IF Handle& <> 0 THEN CALL xClose&(Handle&)
        IF Buffer& <> 0 THEN CALL FreeMem&(Buffer&,BufferSize&)
        CALL DialogBox(Dialog$,1,True,x1b%,y1b%,x2b%,y2b%,False)
        CALL DoDialog(n%,1,-1,-1,-1,-1,x1b%,y1b%,x2b%,y2b%,-1,-1,-1,-1)
    END IF
    CLS
    GOSUB MakeMenu
RETURN
```

Und wenn Sie wissen wollen, wie viele freie und schon belegte Elemente in dem 'Array K()' (Objektdefinitionen) enthalten sind, brauchen Sie nur 'Info' aufzurufen, genauso wie Sie nur 'Help' aufrufen müssen, um zu erfahren, wie das Programm über Tastatur gesteuert werden kann. (Welche Taste für welchen Menüpunkt gedrückt werden muß, ist auch in den Menüzeilen angegeben. Doch zum Menüaufbau später.)

```
Info:
  ' Information ueber freie Elemente von K()
  Status$ = " Status: Info"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  CALL Ausgabe("3D - CAD",60,False)
  CALL Ausgabe("Original-Version: Peter Schulz (c) 1986",75,False)
  CALL Ausgabe("Amiga-Version: Bruno Jennrich (c) 1987",90,False)

  a$ = "Maximale Objektanzahl : "+STR$(MaxAnzahl)
  CALL Ausgabe(a$,105,False)

  a$ = "Aktuelle Objektanzahl : "+STR$(AnzahlK)
  CALL Ausgabe(a$,120,False)

  GOSUB Warte

  CLS
  GOSUB MakeMenu
RETURN


Help:
  ' Welcher Tastendruck fuer welche Aktion
  Status$ = " Status: Help"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  LOCATE 2,1

  PRINT " P rojektionspunkt"
  PRINT " H auptpunkt"

  PRINT " W inkel eingeben"
  PRINT " Cursortasten => drehen"
  PRINT " R otieren (SHIFT,CONTROL)"

  PRINT " D: Abstand zum Hauptpunkt"
  PRINT " +|-|*|/ Abstand vergrößern/-kleinern (SHIFT+|-)"

  PRINT " V ergrößern"
```

```
    PRINT " L ade Objekte"
    PRINT " S peichere Objekte"
    PRINT " M erge Objekte"
    PRINT " N ew, alle Objekte löschen"
    PRINT " B ild abspeichern"
    PRINT " Q => Programmende (Quit)"

    PRINT " F1  => Editor aufrufen"

    PRINT " F9  => Schattieren initialisieren"
    PRINT " F10 => Schattieren"
    PRINT " <SPACE> => Bild zeigen"
    PRINT " C lear screen";

    GOSUB Warte
    CLS
    GOSUB MakeMenu
  RETURN
```

Bevor wir zu den "System"-Routinen kommen, wollen wir Ihnen nun noch die Routine, die für das Vergrößern zuständig ist, vorstellen:

```
  Vergroessern:
    Status$ = " Status: Bildausschnitt vergrößern"+CHR$(0)
    CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

    GOSUB DeleteMenu

    CALL DialogBox("Proportional",0,True,x1a%,y1a%,x2a%,y2a%,False)
    CALL DialogBox("Verzerrt",2,False,x1b%,y1b%,x2b%,y2b%,False)

    CALL DoDialog(n%,0,x1a%,y1a%,x2a%,y2a%,x1b%,y1b%,x2b%,y2b%,-
    1,-1,-1,-1)
    CLS

    IF n%=0 THEN
      LOCATE 10,1
      PRINT " Vergrößerungsfaktor: ";
      a=1
      CALL FormInput (a,30!,.1,100!)
```

```
      CLS
      Bildbreite=a*FaktorX
      Bildhoehe=a*FaktorY
      Bildx=RasterW2%/Bildbreite
      Bildy=RasterH2%/Bildhoehe
    ELSE
      WartFlg! = False
      GOSUB ZeichneNeu
      CALL Rubberbox (Sx%,Sy%,Breite%,Hoehe%,False)
      Bildbreite=RasterW%/Breite% * FaktorX
      Bildhoehe=RasterH%/Hoehe% * FaktorY
      Bildx=(RasterW2%-Sx%)/FaktorX
      Bildy=(RasterH2%-Sy%)/FaktorY
    END IF
    Neu! = True
    WartFlg! = True
    GOSUB ZeichneNeu
  RETURN
```

In dieser Routine werden 'BildX' und 'BildY' je nach Vergrößerungsart (Proportional oder Verzerrt) neu berechnet. Doch nun zu den "System"-Routinen: Was ist typisch für den Amiga? Die Menüsteuerung. Jedes Programm macht von diesen Menüs Gebrauch, so auch unseres. Zum Aufbau und zur Abfrage der Menüs stehen diese Routinen zur Verfügung:

```
  MakeMenu:
    Status$ = " Status: Menu aufbauen"+CHR$(0)
    CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

    MENU 1,0,1, " Amiga"
      MENU 1,1,1," 3-D-Cad-Info  I"

    MENU 2,0,1,"Datei"
      MENU 2,1,1,         " Laden           L"
      MENU 2,2,Start%,    " Speichern       S"
      MENU 2,3,1,         " Mergen          M"
      MENU 2,4,1,         " New             N"
      MENU 2,5,0,         "-----------------"
      MENU 2,6,1,         " Material laden  "
      MENU 2,7,0,         "-----------------"
      MENU 2,8,Start%,    " Hintergrund     "
      MENU 2,9,Start%,    " Bild speichern B"
```

```
    MENU 2,10,Start%,  " Hardcopy         "
    MENU 2,11,0,       "-----------------"
    MENU 2,12,Start%,  " Farbpalette     F"
    MENU 2,13,0,       "-----------------"
    MENU 2,14,1,       " Programmende    Q"

  MENU 3,0,1,"Editor"
    MENU 3,1,1, " Editor aufrufen  ^F1"

  MENU 4,0,Start%,"Parameter"
    MENU 4,1,Start%, " Projektionspunkt  P"
    MENU 4,2,Start%, " Hauptpunkt        H"
    MENU 4,3,Start%, " a, ß, c           W"
    MENU 4,4,Start%, " Abstand           D"
    MENU 4,5,0,      "-------------------"
    MENU 4,6,Start%, " Vergrößern        V"
    MENU 4,7,Start%, " Anzahl Ecken      E"

  MENU 5,0,Start%,"Zeichne"
    MENU 5,1,Start%," Schattieren     F10"
    MENU 5,2,Start%," initialisieren  F9 "
    MENU 5,3,0,     "-------------------"
    MENU 5,4,Start%," Drahtmodell     < >"
    MENU 5,5,Start%," Clear screen    C  "

  GOSUB NOOP
  CALL EmptyBuffers
RETURN


MessageEvent:
  MTitel = MENU(0)
  MPoint = MENU(1)

  IF MTitel <> 0 THEN

    IF MTitel = 1 THEN
      IF MPoint = 1 THEN GOSUB Info
      GOTO MessageEnde
    END IF

    IF MTitel = 2 THEN
      IF MPoint = 1 THEN GOSUB Laden
      IF MPoint = 2 THEN GOSUB Speichern
      IF MPoint = 3 THEN GOSUB Mergen
      IF MPoint = 4 THEN GOSUB FeldInit
```

```
      ' Mpoint = 5 : "---------"
      IF MPoint = 6 THEN GOSUB LoadMat
      ' MPoint = 7 : "---------"
      IF MPoint = 8 THEN GOSUB Hintergrund
      IF MPoint = 9 THEN GOSUB BildSpeichern
      IF MPoint = 10 THEN GOSUB HardCopy
      ' MPoint = 11 : "---------"
      IF MPoint = 12 THEN GOSUB FarbPalette
      ' MPoint = 13 : "---------"
      IF MPoint = 14 THEN GOSUB Quit
      GOTO MessageEnde
    END IF

    IF MTitel = 3 THEN
      IF MPoint = 1 THEN GOSUB EditorAufrufen
      GOTO MessageEnde
    END IF

    IF MTitel = 4 THEN
      IF MPoint = 1 THEN GOSUB InputP
      IF MPoint = 2 THEN GOSUB InputH
      IF MPoint = 3 THEN GOSUB InputWinkel
      IF MPoint = 4 THEN GOSUB InputDPH
      ' MPoint =5: "----------"
      IF MPoint = 6 THEN GOSUB Vergroessern
      IF MPoint = 7 THEN GOSUB WievielEcken
      GOTO MessageEnde
    END IF

    IF MTitel = 5 THEN
      IF MPoint = 1 THEN GOSUB Schattiere
      IF MPoint = 2 THEN GOSUB InitParameters
      IF MPoint = 4 THEN GOSUB ZeichneNeu
      IF MPoint = 5 THEN GOSUB BildLoeschen
    END IF
  END IF
  MessageEnde:
RETURN
```

Wurde ein Menüpunkt angewählt, so muß dafür gesorgt werden, daß alle Menüpnukte deaktiviert werden, da man sonst durch Zufall einen weiteren Menüpunkt auswählen könnte, obwohl man dies gar nicht wollte. Die Menüanfragen werden nämlich

intern aufgelistet und nacheinander abgearbeitet - keine Anfrage geht verloren. Schaltet man die Menüs aber ab, so werden zufällige Menüauswahlen nicht mehr bearbeitet.

```
DeleteMenu:
  MENU 1,0,0,"Amiga"
  MENU 2,0,0,"Datei"
  MENU 3,0,0,"Editor"
  MENU 4,0,0,"Parameter"
  MENU 5,0,0,"Zeichne"
RETURN
```

Will man wieder in die Hauptschleife zurück, in der die Menüabfragen beantwortet werden, muß man natürlich dafür sorgen, daß die Menüs wieder aktiviert werden (s. MakeMenu).

Neben der Menüsteuerung kann man das Programm aber auch über 'Shortcuts', also Tastendrücke, steuern (s. 'Help'). Für die meisten Menüpunkte stehen diese Shortcuts in der jeweiligen Menüzeile. Da es aber auch Funktionen gibt, die nur von Tastatur aus ausgeführt werden können, haben wir in 'Help' dafür gesorgt, daß die wichtigsten 'Shortcuts' aufgelistet werden ('Help' wird entweder durch die Help-Taste oder das Fragezeichen aufgerufen).

Wartet das Programm auf Ihre Eingaben, führt es also gerade keine Schattierberechnungen o.ä. durch, können Sie dies an dem Status 'NOP' für 'No Operation' erkennen. In der Routine 'NOOP' wird dieser Status in die oberste Window-Zeile geschrieben.

```
NOOP:
  Status$ = " Status: NOP"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
RETURN
```

Für die Steuerung von Dialogboxen (eine Art Requester) haben wir eigene Routinen geschrieben: In 'DialogBox' werden diese aufgebaut, und 'DoDialog' wartet dann auf ein Auswahl einer DialogBox.

```
SUB DialogBox(Dialog$,Position%,Ret,x1%,y1%,x2%,y2%,Scr) STATICSHARED
RastPort&,NRastPort&,True,RasterW1%,RasterW2%
  ' Eigene DialogBox erzeugen
  ' Dialog$ = Text in DialogBox
  ' Position%
  '    0 = Rechts
  '    1 = Mitte
  '    2 = Links
  ' Ret% = Bestaetigung durch Return möglich?
  '    True  = Ja
  '    False = Nein
  ' x1%,y1%,x2%,y1% = Groesse des 'Klick'-Feldes (wird zurueckgegeben)
  '       Scr = In welchem Screen ausgeben?
  '      True = Display-Screen
  '     False = Benutzer-Screen

 IF Scr = True THEN
   rp& = RastPort&
   Middle& = RasterW2%
     SEnd& = RasterW1%
  '  Wir schreiben in Display-Screen
 ELSE
   rp& = NRastPort&
   Middle& = 320
     SEnd& = 640
  '  RastPort des Basic Windows
 END IF

   y1% = 150
   y2% = 150 + 6 + 10

   Length& = TextLength&(rp&,SADD(Dialog$),LEN(Dialog$))

 IF Position% = 0 THEN
   x1% = 70
   x2% = x1% + Length& + 20
 END IF

 IF Position% = 1 THEN
   x1% = Middle&-Length&/2-10
   x2% = Middle&+Length&/2+10
 END IF
```

```
  IF Position% = 2 THEN
    x1% = SEnd&-70-20-10-Length&
    x2% = SEnd&-70-10
  END IF

  CALL Move&(rp&,x1%+10,y1%+10)
  CALL Text&(rp&,SADD(Dialog$),LEN(Dialog$))

  IF Scr = True THEN
    CALL Box(rp&,x1%,y1%,x2%,y2%)
    CALL Box(rp&,x1%-4,y1%-2,x2%+4,y2%+2)
  IF Ret = True THEN CALL Box(rp&,x1%-2,y1%-1,x2%+2,y2%+1)
  ELSE
    LINE (x1%,y1%)-(x2%,y2%),,b
    LINE (x1%-2,y1%-2)-(x2%+2,y2%+2),,b
  IF Ret = True THEN LINE (x1%-1,y1%-1)-(x2%+1,y2%+1),,b
  END IF
END SUB

SUB
DoDialog(n%,Ret%,x1a%,y1a%,x2a%,y2a%,x1b%,y1b%,x2b%,y2b%,x1c%,y1c%,x2c%,
y2c%) STATIC
  ' Dialogboxen abfragen
  ' n%: welche Box angeklickt?
  ' Ret%: welche Box kann mit <Return> bestätigt werden?
  ' x1.,y1.,x2.,y2. Koordinaten der Boxen
  CALL EmptyBuffers
  n% = -1
  WHILE (n% = -1)
    IF MOUSE(0) <> 0 THEN
      x = MOUSE(1)
      y = MOUSE(2)

      IF (x1a%<x) AND (x2a%>x) AND (y1a%<y) AND (y2a%>y) THEN n% = 0
      IF (x1b%<x) AND (x2b%>x) AND (y1b%<y) AND (y2b%>y) THEN n% = 1
      IF (x1c%<x) AND (x2c%>x) AND (y1c%<y) AND (y2c%>y) THEN n% = 2
    END IF
    IF INKEY$ = CHR$(13) THEN n% = Ret%
  WEND
END SUB
```

Zur Ausgabe von Texten haben wir die Routine 'Ausgabe' geschrieben:

```
SUB Ausgabe(Aus$,y%,Scr) STATIC
SHARED RastPort&,NRastPort&,RasterW2%,True,Length&
  ' Text Mittenzentriert auf einen der beiden Screens ausgeben
  ' Aus$: Ausgabestring
  ' y%: vertikale Position?
  ' Scr: Benutzer oder Display Screen?
  IF Scr = True THEN
    rp& = RastPort&
    Middle& = RasterW2%
  '   Wir schreiben in Neuen (Display) Screen
  ELSE
    rp& = NRastPort&
    Middle& = 320      ' Benutzer-Screen (NOOP Window)
  END IF
  Length& = TextLength&(rp&,SADD(Aus$),LEN(Aus$))
  Middle& = Middle& - Length&/2
  CALL Move& (rp&,Middle&,CLNG(y%))
  CALL Text& (rp&,SADD(Aus$),LEN(Aus$))
END SUB
```

Der auszugebende Text wird dabei zentriert ausgegeben. Dies ist besonders als Informationszeile über Dialogboxen sinnvoll.

Damit aber eine Box, also ein Rechteck, um unsere Texte der Dialogboxen gezeichnet wird, wird weiterhin die Routine 'Box' aufgerufen, die auch von 'RubberBox' zur Festlegung eines Bildschirmausschnittes mit der Maus benutzt wird.

```
SUB Box(RastPort&,x1%,y1%,x2%,y2%) STATIC
  ' Rechteck in Display-Screen
  ' RastPort&: in welche RastPort zeichnen
  ' x1,y1,x2,y2 Koordinaten des Rechtecks
  CALL Move&(RastPort&,x1%,y1%)
  CALL Draw&(RastPort&,x2%,y1%)
  CALL Draw&(RastPort&,x2%,y2%)
  CALL Draw&(RastPort&,x1%,y2%)
  CALL Draw&(RastPort&,x1%,y1%+1)
END SUB
```

```
SUB Rubberbox( x%, y%, w%, h%, Back!) STATIC
SHARED
RasterW1%,RasterH1%,WScreen&,NRastPort&,RastPort&,NWScreen&,NWBase&,
WBase&,Length&,True,False
  ' Rechteck mit Maus festlegen
  ' x%,y%: linke obere Ecke
  ' w%,h%: Breite, Höhe
  ' Back!: zurueck zum Benutzer Fenster?
  CALL Scron
  CALL SetDrMd&(RastPort&,2)  'COMPLEMENT

  CALL SetAPen&(RastPort&,1)

 Repitition:
  Oldx% = 1
  OldY% = 1
  Flag! = 0

  CALL EmptyBuffers

   mouse1:
    x% = PEEKW(WScreen&+18)
    y% = PEEKW(WScreen&+16)

      IF (x% <> Oldx%) OR (y%<>OldY%) THEN
        IF Flag! = 1 THEN
          CALL Move&(RastPort&,Oldx%,0)
          CALL Draw&(RastPort&,Oldx%,RasterH1%)
          CALL Move&(RastPort&,0,OldY%)
          CALL Draw&(RastPort&,RasterW1%,OldY%)
        END IF

        Flag! = 1

        CALL Move&(RastPort&,x%,0)
        CALL Draw&(RastPort&,x%,RasterH1%)
        CALL Move&(RastPort&,0,y%)
        CALL Draw&(RastPort&,RasterW1%,y%)
        Oldx% = x%
        OldY% = y%
      END IF

   IF MOUSE(0) = 0 GOTO mouse1

   CALL Move&(RastPort&,x%,0)
   CALL Draw&(RastPort&,x%,RasterH1%)
```

```
CALL Move&(RastPort&,0,y%)
CALL Draw&(RastPort&,RasterW1%,y%)

Oldx% = -1
OldY% = -1
Flag! = 0

WHILE MOUSE(0) <> 0

  w% = PEEKW(WScreen&+18)
  h% = PEEKW(WScreen&+16)
  IF (w% <> Oldx%) OR (h% <> OldY%) THEN

    IF Flag! = 1 THEN CALL Box(RastPort&,x%,y%,Oldx%,OldY%)
    Flag! = 1

      IF (w% < x%+6) THEN w% = x%+6
      IF (h% < y%+3) THEN h% = y%+3

      IF (w% > x%) AND (h% > y%) THEN
        IF w%>RasterW1% THEN w% = RasterW1%
        IF h%>RasterH1% THEN h% = RasterH1%
        Oldx% = w%
        OldY% = h%

        CALL Box(RastPort&,x%,y%,w%,h%)
      END IF
    END IF
  WEND

  IF Flag = 1 THEN CALL Box(RastPort&,x%,y%,w%,h%)

  CALL Ausgabe("Ausschnitt Ok ?",130,True)

  Flag! = 0
  Oldx% = -1
  OldY% = -1

  CALL DialogBox("OK",1,True,x1a%,y1a%,x2a%,y2a%,True)

  CALL EmptyBuffers

  a$ = ""
  WHILE (MOUSE(0) = 0) AND (a$ <> CHR$(13))
    a$ = INKEY$
  WEND
```

```
        Mx% = PEEKW(WScreen&+18)
        My% = PEEKW(WScreen&+16)

        Nochmal = True
        IF (Mx%>x1a%) AND (Mx%<x2a%) AND (My%>y1a%) AND (My%<y2a%) THEN
        Nochmal = False
        IF a$ = CHR$(13) THEN Nochmal = False

        CALL Ausgabe("Ausschnitt Ok ?",130,True)

        CALL DialogBox("OK",1,True,x1a%,y1a%,x2a%,y2a%,True)

    IF Nochmal = True THEN GOTO Repitition

    w% = w%-x%
    h% = h%-y%

    CALL SetDrMd&(RastPort&,1)  'JAM2
    CALL SetDrMd&(NRastPort&,1)

    IF Back! = True THEN CALL Scroff
END SUB
```

Wir haben deshalb unsere eigene 'Box'-Routine benutzt und auf die Unterart des LINE-Befehls verzichtet, weil wir Dialogboxen sowohl im Display-Screen als auch im Benutzer-Screen verwenden wollen.

Dies allein ist aber noch kein Hindernis für die LINE-Alternative. Da wir aber direkt auf den Rastport unseres Display-Screens zugreifen und diesen auch verändern (z.B. den Zeichenmodus), um auf die gesamte Bitmap zuzugreifen, ist LINE nicht immer möglich. (Wir verwenden auch die Punkte, die normalerweise die Umrandung eines Windows oder Screens darstellen.)

Um auf einen Tastendruck der Tastatur oder Maus zu warten, haben wir die Routine 'Warte' geschrieben. Nachdem z.B. das Unterverzeichnis der Diskette in 'Directory' aufgelistet wurde

oder der 'Info'- bzw. 'Help'-Text ausgegeben wurde, wird auf solch einen Tastendruck gewartet, bevor in die Hauptschleife zurückgekehrt wird.

Vorher wird allerdings der Maus- und Tastaturbuffer geleert. So werden alle Tastendrücke vor der Abfrage ignoriert, und es kann auf die tatsächlich neuen Tastendrücke reagiert werden.

```
Warte:
  ' Warte auf (Maus-)Tastendruck
  CALL EmptyBuffers                           ' Buffer leeren
  WHILE (INKEY$ = "") AND (MOUSE (0) = 0)
  WEND
RETURN


SUB EmptyBuffers STATIC
  ' Maus- und Tastaturbuffer leeren
  WHILE MOUSE(0) <> 0
  WEND
  WHILE (INKEY$<>"")
  WEND
END SUB
```

Für eine Eingabe ganz anderer Art sind folgende Routinen zuständig:

```
SUB FormInput (i,laenge,unt!,ob!) STATIC
'Eingabe eines einzigen Realwert
'i =>            einzulesende Zahlvariable (alter Wert wird
 ausgegeben)
'laenge,unt!,ob! => s.o.
  z=CSRLIN
  s=POS(0)
  a$ = " "
  CALL PutReal (i,z,s,laenge)
  CALL GetReal (i,a$,z,s,laenge,unt!,ob!)
END SUB
```

```
SUB FormInputInt (i,laenge,unt!,ob!) STATIC
'Eingabe eines einzigen Integerwertes
'i =>             einzulesende Zahlvariable (alter Wert wird
 ausgegeben)
'laenge,unt!,ob! => s.o.
  z=CSRLIN
  s=POS(0)
  a$ = " "
  CALL PutReal (i,z,s,laenge)
  CALL GetInt (i,a$,z,s,laenge,unt!,ob!)
END SUB


SUB FormInputString (s$,laenge) STATIC
' String einlesen (z.B. Filename)
  a$ = " "
  s = POS(0)
  z = CSRLIN
  CALL PutString (s$,z,s,laenge)
  CALL GetString (s$,a$,"abcdefghijklmnopqrstuvwxyzäöüß
  ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ1234567890.-_:/",z,s,laenge)
END SUB
```

Diese Routinen erwarten die Eingabe eines Real-Wertes, einer 'Integer'-Zahl oder einer Zeichenkette. Dabei werden die Eingabe-Routinen des Editors benutzt.

Nicht zugelassene Zeichen, also z.B. Buchstaben bei der Eingabe von Zahlen, werden ignoriert. Geben Sie jedoch Werte an, die kleiner als die angegebene Untergrenze oder größer als die Obergrenze sind, so blitzt der Bildschirm kurz auf.

Bei allen Floppy-Computer-Operationen wird Ihnen vorher die Möglichkeit gegeben, das aktuelle Unterverzeichnis zu betrachten oder ein neues Unterverzeichnis auszuwählen. Die Routine 'Directory' macht es möglich:

```
Directory:
  ' Directory sehen oder Drive ändern
  Status$ = " Status: Directory"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)
```

```
    GOSUB DeleteMenu
    a$ = "Aktuelles Directory: "+AktDrive$
    CALL Ausgabe(a$,50,False)
    CHDIR AktDrive$

    RepeatD:
      CALL DialogBox("Files",0,False,x1a%,y1a%,x2a%,y2a%,False)
      CALL DialogBox("Weiter",1,True,x1b%,y1b%,x2b%,y2b%,False)
      CALL DialogBox("Chdir",2,False,x1c%,y1c%,x2c%,y2c%,False)

      CALL DoDialog(n%,1,x1a%,y1a%,x2a%,y2a%,x1b%,y1b%,x2b%,y2b%,
      x1c%,y1c%,x2c%,y2c%)

      CLS
      IF n% = 0 THEN
        FILES
        GOSUB Warte
        CLS
        GOTO RepeatD
      END IF
      IF n% = 2 THEN
        a$ = "Aktuelles Directory: "+AktDrive$
        CALL Ausgabe(a$,50,False)

        LOCATE 10,1
        PRINT "        Neues Directory: ";
        CALL FormInputString (AktDrive$,30!)

        CHDIR AktDrive$

        CLS
        GOTO RepeatD
      END IF
    CLS
  RETURN
```

Die Neuwahl eines Diskettenverzeichnisses ermöglicht aber auch, andere Disketten zu benutzen. Vielleicht kennen Sie das Dilemma, in dem sich jeder Amiga-Benutzer befindet, wenn plötzlich ein Diskettenwechsel vorgenommen werden muß (z.B. wegen eines Lesefehlers), aber kein 'chdir' oder 'cd' ausgeführt werden kann, und es unmöglich wird, eine neue Diskette zu benutzen. Unter Umständen kann es so zu Datenverlusten kom-

men. Nicht aber bei unserem Programm. Mit 'Directory' können Sie nämlich wie gesagt auch neue Disketten benutzen, ohne das Programm unterbrechen zu müssen.

Wollen Sie das Programm verlassen, wird 'Quit' aufgerufen. In dieser Routine werden Sie aber noch einmal über Ihren Entschluß, daß Programm beenden zu wollen, befragt und können diesen gegebenfalls widerrufen. Wollen Sie das Programm tatsächlich verlassen, so wird vorher noch der gesamte belegte Speicher freigegeben und alle Libraries und der neu eröffnete Display-Screen geschlossen:

```
CloseIt:
  ' Alles schließen und Speicher FREEn
  GOSUB CloseGfx
  WINDOW CLOSE 2
  SCREEN CLOSE 1      ' evt. belegte 6. BitMap wird automatisch
  MENU RESET          ' freigegeben

  IF SetPointAdr& <> 0 THEN CALL FreeMem&(SetPointAdr&,SetPointAnz&)
  IF MusterS& <> 0 THEN CALL FreeMem&(MusterS&,(AnzahlMusterS%+1)*2*16)
  IF MusterX& <> 0 THEN CALL FreeMem&(MusterX&,(AnzahlMusterX%*2+1)*
  2*16)
    ' Speicher der Muster und von ASSEM-Routine freigeben
  LIBRARY CLOSE
RETURN


Quit:
  CALL Scroff
  Status$ = " Programmende?"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  CALL Ausgabe("Wollen Sie das Programm wirklich verlassen?",100,False)

  CALL DialogBox("Nein",0,True,x1a%,y1a%,x2a%,y2a%,False)
  CALL DialogBox("Neustart",1,False,x1b%,y1b%,x2b%,y2b%,False)
  CALL DialogBox("Ja",2,False,x1c%,y1c%,x2c%,y2c%,False)

  CALL DoDialog(n%,0,x1a%,y1a%,x2a%,y2a%,x1b%,y1b%,x2b%,y2b%,x1c%,
  y1c%,x2c%,y2c%)
```

```
  CLS
  IF n% > 0 THEN
    GOSUB CloseIt
    IF n% = 1 THEN
      RUN
    ELSE
      END
    END IF
  END IF
  GOSUB MakeMenu
RETURN
```

Weiterhin haben Sie die Möglichkeit, einen Neustart zu veranlassen, anstatt das Programm endgültig zu beenden.

Um zwischen dem neuen 'Display'-Screen und dem Benutzer-Screen hin und her schalten zu können, gibt es 'Scron' und 'Scroff' (Display-Screen 'an' und 'aus'). Um den Display-Screen zu löschen - z.B. vor dem Neuzeichnen des Drahtmodells -, wird 'BildLoeschen' aufgerufen.

```
SUB Scron STATIC
SHARED WScreen&,WBase&
  ' Display Screen an, Benutzer Screen aus
  CALL ScreenToFront&(WScreen&)
  WINDOW OUTPUT 2
  WINDOW 2
  CALL ActivateWindow&(WBase&)
END SUB


SUB Scroff STATIC
SHARED NWBase&,NWScreen&
  ' Display Screen aus , Benutzer Screen an
  CALL ScreenToFront&(NWScreen&)
  WINDOW OUTPUT 1
  WINDOW 1
  CALL ActivateWindow&(NWBase&)
END SUB
```

```
BildLoeschen:
  Status$ = " Status: Clear screen"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  CALL SetRast&(RastPort&,0)
  Hg = True

  GOSUB NOOP
RETURN
```

Mit 'EditorAufrufen' haben Sie dann noch vom Tracer aus die Möglichkeit, den Objekteditor aufzurufen, der dann von Diskette geladen und anschließend ausgeführt wird (CHAIN-Anweisung).

```
EditorAufrufen:
  Status$ = " Status: Editor aufrufen"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  CALL DialogBox("nicht Aufrufen",0,True,x1a%,y1a%,x2a%,y2a%,False)
  CALL DialogBox("Aufrufen",2,False,x1c%,y1c%,x2c%,y2c%,False)
  CALL DoDialog(n%,0,x1a%,y1a%,x2a%,ya2%,-1,-1,-1,-1,x1c%,y1c%,
  x2c%,y2c%)
  CLS
  IF n% = 2 THEN
    CHDIR "Tracer:"
    GOSUB CloseIt
    CHAIN "Editor"
  END IF
  GOSUB MakeMenu
END
```

'Fehlerbehandlung' schließlich sorgt noch dafür, daß Ihnen auftretende Fehler mitgeteilt werden. Diese dürften sich eigentlich nur auf I/O-Errors, also auf Ein-/Ausgabe-Fehler (z.B. Schreib-/Lesefehler auf der Diskette etc.) beschränken. Sollte in Ihrem Programm aber ein anderer Fehler auftreten, so wird Ihnen die Fehlernummer mitgeteilt (z.B '2' für 'SyntaxError', s. Anhang B des BASIC-Handbuchs).

```
FehlerBehandlung:
  Fehler = ERR

  Status$ = " Status: Next Big Error!!!"+CHR$(0)
  CALL SetWindowTitles&(NWBase&,SADD(Status$),0)

  GOSUB DeleteMenu

  Dialog$ = ""
  IF Fehler = 64 THEN         ' Bad File Mode
    Dialog$ = "Ungültiger Dateiname!!!"
  END IF

  IF Fehler = 57 THEN         ' Device I/O Error
    Dialog$ = "Device I/O Error!!!"
  END IF

  IF Fehler = 68 THEN         ' Device unable
    Dialog$ = "Gerät nicht verfügbar"
  END IF

  IF Fehler = 61 THEN         ' Disk Full
    Dialog$ = "Diskette ist voll!!!"
  END IF

  IF Fehler = 53 THEN         ' File not found
    Dialog$ = "File nicht gefunden!!!"
  END IF

  IF Fehler = 70 THEN         ' Permission denied
    Dialog$ = "Schreibschutz entfernen!!!"
  END IF

  IF Dialog$ = "" THEN
    Dialog$ = "Fehler Nummer"+STR$(Fehler)
  END IF

  CALL DialogBox(Dialog$,1,True,x1a%,y1a%,x2a%,y2a%,False)

  CALL DoDialog(n%,1,-1,-1,-1,-1,x1a%,y1a%,x2a%,y2a%,-1,-1,-1,-1)
```

```
  CLS
  GOSUB MakeMenu
  RESUME ok
RETURN
```

4.3.2 Das Hauptprogramm aufbauen

Um nun, nachdem alle Routinen des Tracers bekannt sind, das Hauptprogramm zu erzeugen, müssen Sie nur die auf der Diskette enthaltenen Programmmodule, die als ASCII-Files abgespeichert wurden, miteinander verbinden (Mergen).

Dazu geben Sie nach dem Start von AmigaBASIC erst einmal 'clear ,140000' ein. Hiermit wird der nötige Programmspeicher reserviert (Diesen Befehl müssen Sie vor jedem weiteren Start des Tracers ausführen. Allerdings funktioniert das nur bei Amigas mit mindestens 512KByte. Aber selbst dann müssen Sie das AmigaDOS-Window so klein wie möglich machen, um Speicherplatz zu gewinnen, vorausgesetzt, Sie arbeiten mit dem CLI.)

Bitte setzen Sie das aktuelle Verzeichnis auf "tracer:modules". Sollten Sie eine Sicherheitsdiskette der Buchdiskette angefertigt haben, so tragen Sie bitte statt "tracer" den richtigen Namen ein.

Dann brauchen Sie die Module, die sich im Unterverzeichnis 'Modules' auf der mitgelieferten Diskette befinden, nur noch mit folgenden Befehlen, die im Direktmodus eingegeben werden, zusammenmergen:

```
merge "Init"
merge "System"
merge "DrahtMod-Zeichnen"
merge "DrahtMod-Eingabe"
merge "Schatt-Init"
merge "Schattieren"
merge "Service"
```

Danach sollten Sie das Programm unbedingt abspeichern:

```
save "tracer:tracer".
```

Sie können dieses Programm entweder auf die mitgelieferte Diskette schreiben oder eine eigene Diskette benutzen - z.B. um die Originaldiskette als Sicherheitskopie zu verwahren.

Die eigene Diskette muß dann aber den Namen 'Tracer' tragen (format drive df0: name "Tracer"). Weiterhin müssen das Programm, der Editor und die Maschinenroutine 'SetPoint.B' im Hauptverzeichnis der Diskette abgespeichert worden sein.

Letztlich muß dann nur noch dafür gesorgt werden, daß die '.bmap'-Files, die unsere Programme benötigen, auch auf der neuen Diskette sind ("dos.bmap", "exec.bmap", "graphics.bmap", "intuition.bmap"). Dazu kopieren Sie das Unterverzeichnis 'Libs' der Originaldiskette auf das von Ihnen angelegte Unterverzeichis 'Libs' (Makedir Libs) der neuen Diskette.

Am einfachsten ist es natürlich, wenn Sie zuvor mittels Diskcopy eine Kopie der gesamten Diskette erstellen. Nach dem Zusammenmergen der einzelnen Module können Sie diese dann aus dem Modules-Verzeichnis löschen, um mehr Platz auf der Diskette zu haben. Tun Sie dies aber nur mit Ihrer Kopie und keinesfalls mit dem Original! Nach dem Abspeichern des kompletten Tracers befindet sich das vollständige Programm auf Diskette und steht für Sie bereit zu einer Entdeckungsfahrt in die dreidimensionale Computer-Welt!

5. Die Bedienung der Programme

Auch wenn Sie ja praktisch mit uns zusammen die Programmodule erstellt haben, vergißt man doch leicht die Bedienung. Deshalb haben wir im folgenden Kapitel die Bedienung der Programme noch einmal ausführlich erläutert.

5.1 Handbuch zum Editor

5.1.1 Allgemeines zur Eingabe

Sämtliche Eingaben erfolgen in unsichtbaren Fenstern. Innerhalb dieses Fensters kann die Eingabe mit den Cursor-Tasten, Delete und Backspace editiert werden, wobei immer der Insert-Modus verwendet wird. Ist die Eingabe länger als das Fenster, so wird sie nach links verschoben, und die Anfangszeichen verschwinden, bleiben jedoch existent. Mit Hilfe von Cursor-Left oder -Right können Sie den sichtbaren Ausschnitt bestimmen. Zu beachten ist weiterhin, daß je nach gewünschter Eingabe nur bestimmte Tasten vom Programm akzeptiert werden. Neben den Standard-Tasten werden akzeptiert:

```
a) bei Integer-Werten:     0,1,..,9,-
b) bei Real-Werten:        0,1,..,9,-,.
c) bei Vektoren:           0,1,..,9,-,.,","
d) bei Zeichenketten:      fast alle Zeichen
e) bei Characters:
     - bei der Sichtbar-Eingabe:                       j,n
     - bei der Vergrößerung, Rotation, Verschiebung:   q,c,d
     - bei der Show-Funktion:                          d,m,f
```

Jede Eingabe wird beendet durch ein Return, unabhänig davon, wo sich der Cursor befindet. Akzeptiert wird jeweils die gesamte Eingabe. Nach Characters muß nicht ein Return eingegeben werden, da das Programm nur auf die erste gültige Taste wartet. Bei Werten zwischen Null und Eins muß eine Null vor dem Komma stehen. Erfüllt eine Eingabe nicht die verlangten Formen, so wird sie nicht akzeptiert, das heißt, daß der Cursor

nach Return wieder am Anfang des Strings auftaucht und die Eingabe korrigiert werden muß. Dies ist auch dann der Fall, wenn die eingegebenen Werte nicht im jeweiligen Wertebereich liegen. Dieser Wertebereich ist fast immer [-10000..10000], bis auf folgende vier Ausnahmen:

```
a) bei der Materialart:              1..10000
b) bei dem Innen- oder Außenradius:  0..1
c) bei dem Vergrößerungsfaktor:      1/64..64
d) bei der Linieneinstellung:        1..50
```

Wird in einer Eingabeposition ein Koordinatenvektor verlangt, was in unserem Fall fast immer der Fall ist, so stehen Ihnen zwei Eingabemöglichkeiten zur Verfügung: die konventionelle Eingabe über die Tastatur oder die Eingabe über die Maus. Soll die Eingabe über die Maus erfolgen, so muß der Cursor am Anfang der entsprechenden Eingabezeile stehen, und es darf noch keine gültige Taste bei dieser Eingabe gedrückt worden sein. Mittels der Maus wird dann in den Fenstern die gewünschte Position angeklickt, worauf dann die zugehörigen Koordinaten im Ein- und Ausgabefenster unter dem Stichwort Maus angezeigt werden. Steht dort schließlich die Position, die Sie wollen, so brauchen Sie nur Return zu drücken. Dann wird sofort die Mausposition in die Eingabezeile übernommen, entweder direkt oder als Differenz zwischen dem Mausvektor und dem Bezugsvektor, je nachdem, welche Daten gerade vom Programm gewünscht werden.

Bei der Menüeingabe sind noch einige Besonderheiten zu beachten. Sollten Sie zum Beispiel bei der fünften Eingabeposition feststellen, daß die erste Eingabe fehlerhaft ist, so brauchen Sie nicht noch Mal von vorne anfangen, sondern wandern zum entsprechenden Menüpunkt zurück. Dies geschieht mit der Cursor-Up-Taste. Diese ist nur dann aktiv, wenn in der momentanen Eingabezeile noch keine Taste gedrückt wurde. In der fehlerhaften Zeile brauchen Sie auch nicht alle Zeichen neu eingeben, sondern es reicht, wenn Sie mit den CURSOR dort hin bewegen, wo der Fehler ist, und Sie diesen dann korrigieren. Ist

der Fehler berichtigt, so wird Return gedrückt, und mit Cursor-Down gelangen Sie dann wieder hinunter in die alte Eingabezeile.

5.1.2 Die Eingabe der Körperdaten

Werden die oben beschriebenen programmspezifischen Besonderheiten bei der Eingabe berücksichtigt, so ist die Eingabe der Körperdaten kein Problem mehr, wenn man noch zusätzlich weiß, welche Daten eigentlich gefragt sind und wie die Eingabe eines bestimmten Körpers gestartet wird. Die Informationen zu dem ersten Punkt können Sie den unten stehenden Zeichnungen entnehmen.

Der letztere Punkt ist ebenfalls schnell erklärt: Unter dem Menüpunkt Körper finden Sie alle von uns vordefinierten Körper. Wollen Sie nun einen bestimmten Körper aus dieser Liste eingeben, so braucht er nur in diesem Menü ausgewählt werden. Unmittelbar darauf erscheint im Editor-Fenster ein neuer Datensatz und der Cursor steht an der ersten Eingabeposition. Die Eingabe kann also beginnen. Zu den unten dargestellten Körperdaten werden noch abgefragt, ob der eingegebene Körper anschließend auf dem Bildschirm sichtbar sein soll und welches Material Sie ihm zuordnen wollen, wobei Sie die einzelnen Materialien, die sich in Farbe, Schattenhelligkeit und Spiegelungsfaktor unterscheiden können, durch eine Zahl repräsentieren. Das zu dieser Zahl gehörende Material können Sie in einem weiteren Schritt mit dem Materialeditor eingeben, was weiter unten beschrieben ist.

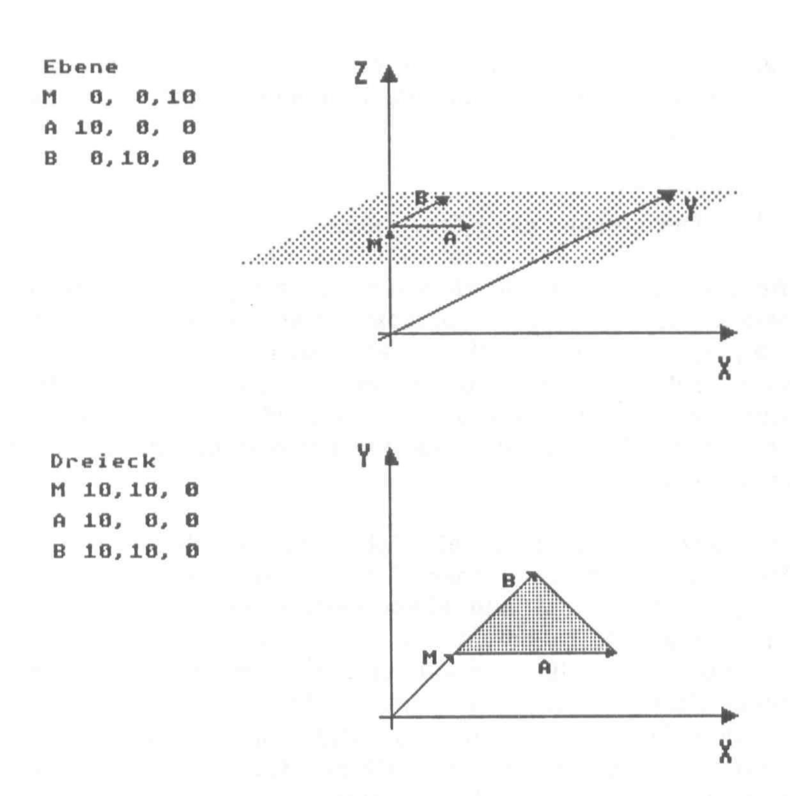
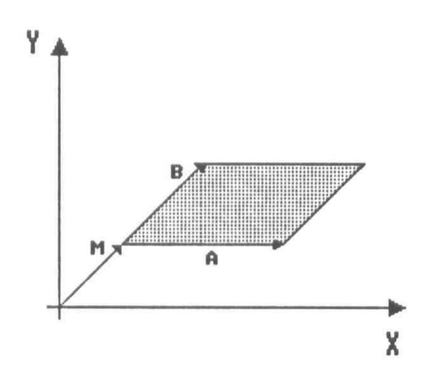


Abbildung 5.1

```
Parallelogramm
M 10,10, 0
A 10, 0, 0
B 10,10, 0
```




```
Kreis
M 10,10, 0
A 10, 0, 0
B  0,10, 0
```

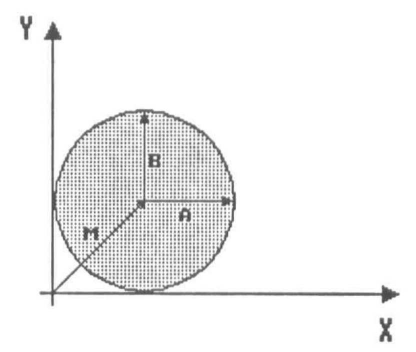
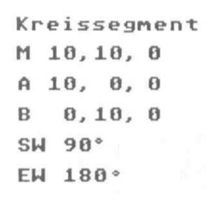
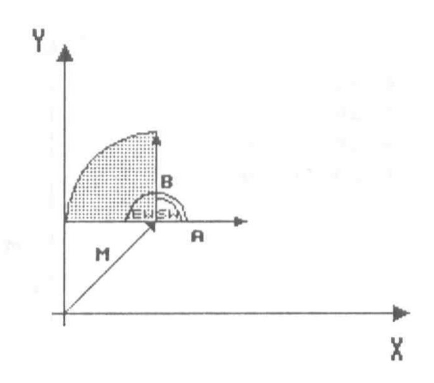


Abbildung 5.2





Kreisteil
M 10,10, 0
A 10, 0, 0
B 0,10, 0
SW 90°
EW 180°
ri 0.6
ra 1

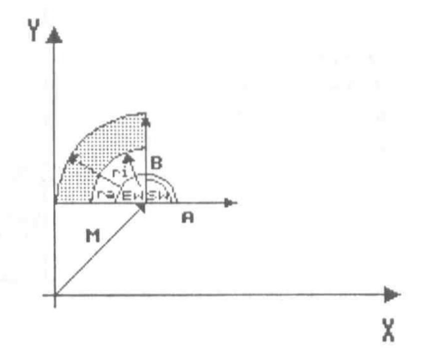


Abbildung 5.3

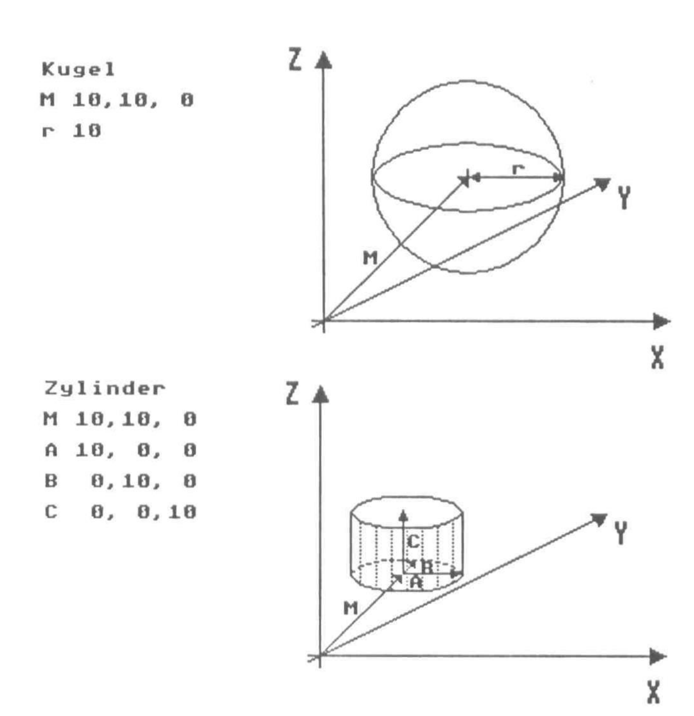


Abbildung 5.4

```
Zylindersegment
M 10,10, 0
A 10, 0, 0
B  0,10, 0
C  0,10, 0
sw 0°
ew 90°
```

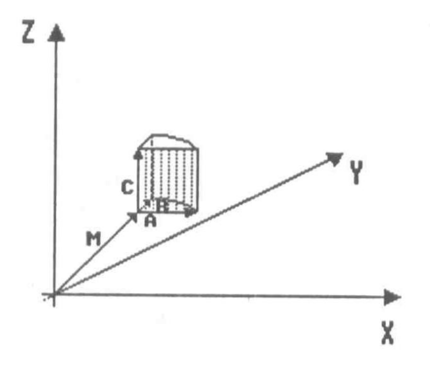
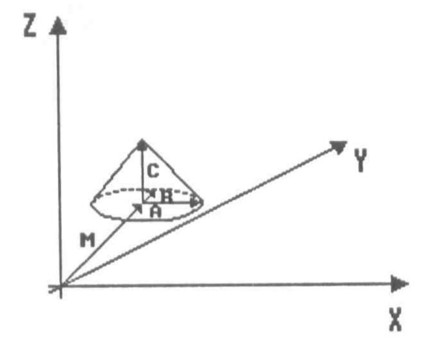


Abbildung 5.5

```
Kegel
M 10,10, 0
A 10, 0, 0
B  0,10, 0
C  0, 0,10
```




```
Sphäroid
M 10,10, 0
A 10, 0, 0
B  0,10, 0
C  0, 0, 5
```

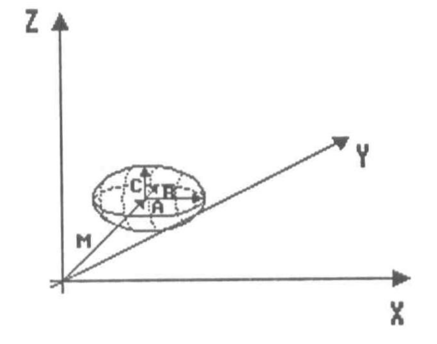


Abbildung 5.6

5.1.3 Die weiteren Operationen

Diese Funktionen stehen alle in dem Menü Operationen. Sie ermöglichen eine Vielzahl von Funktionen bezüglich der Körperliste und der Bildschirmdarstellung, wobei die wichtigsten von ihnen ebenfalls über einen Tastendruck aufgerufen werden können. Mit LINKS kann der Vorgänger des aktuellen (d.h.: des gerade im Editor-Fenster sichbaren) Datensatzes nach vorne geholt werden. Sollten also zum Beispiel die Daten des sechsten Körpers sichtbar sein, so wird nach der LINKS-Funktion i.a. der fünfte Körperdatensatz ausgedruckt. Analog funktioniert die Funktion RECHTS, bloß daß nun der Nachfolger bestimmt wird. Existiert dieser nicht, so wird anschließend wieder das aktuelle Element angezeigt.

Mittels AUSSCHNITT kann der sichtbare Ausschnitt in den Grafik-Fenstern beliebig ausgewählt werden, genauere Informationen hierzu stehen auch in der Programmbeschreibung des Editors. Die drei nun folgenden Funktionen im Menü dienen zur Einstellung des Vergrößerungsfaktors. Mit der ersten wird er verdoppelt, mit der zweiten halbiert, und mit der dritten kann er beliebig innerhalb des Intervalls [1/64,64] gewählt werden. Je größer dieser Vergrößerungsfaktor gewählt wird, desto kleiner wird der sichtbare Ausschnitt, aber um so mehr Details sind erkennbar. Wird der Vergrößerungsfaktor verändert, so bleibt der Bildschirminhalt unverändert, da der Neuaufbau bei großen Bildern viel zu lange dauern würde. Aus diesem Grund ist das Neuzeichnen des Bildschirms Ihnen überlassen, was Sie mittels REFRESH bewerkstelligen können.

Weiteren Einfluß auf die Geschwindigkeit der Grafik haben Sie mit LINIEN. Bei dieser Prozedur können Sie die Anzahl von Linien bestimmen, mit denen ein Kreis gezeichnet wird, jedoch muß dieser Wert zwischen 1 und 50 liegen. Eine weitere Funktion ist SHOW, die Ihnen einen bestimmten Körper zeigt, entweder nur die Daten oder auf dem Bildschirm in grafischer Darstellung, und hierbei entweder in der Mitte von allen drei Fenstern oder in roter Farbe. Welcher Körper und welcher Modus verwendet werden soll, muß natürlich eingegeben werden, wobei d für die Anzeige der Daten und f für die farbliche Her-

vorhebung des Körpers steht. Wird m eingegeben, so wird der Ausschnitt so verschoben, daß der Körper in der Mitte der Fenster zu sehen ist, falls Sie anschließend REFRESH durchführen lassen. Der letzte Punkt des Menüs ist QUIT, mit dessen Hilfe Sie das Programm abbrechen lassen können. Sollte dieser Punkt aus Versehen gewählt werden, so müssen Sie anschließend das Nein-Feld auswählen, andernfalls das Ja-Feld. Jetzt bleibt nur noch eine Funktion übrig, die MAT.EDITOR-Funktion, die den Materialeditor startet.

5.1.4 Der Materialeditor

Wenn Sie die gesamte Körperliste, die die Daten des zu berechnenden Bildes enthält, eingegeben haben, so haben Sie neben allen notwendigen Koordinaten auch jedem Körper eine Materialnummer zugeordnet. Jede dieser Nummern repräsentiert ein spezifisches Material, das Sie bezüglich Farbe, Schattenhelligkeit und Spiegelungsfaktor beliebig selbst zusammensetzen können. Die Materialeingabe erfolgt in dem Materialeditor. Er ordnet jedem Material eine Nummer zu, die mit der gewünschten Materialnummer, die Sie den einzelnen Körpern zugeordnet haben, übereinstimmen muß, damit auch jeder Körper die Farbe erhält, die Sie ihm geben wollen.

Alle Eingaben im Materialeditor erfolgen nicht über Tastatur, sondern über Maus mit Hilfe von Reglern. Wollen Sie mit diesen Reglern einen Wert einstellen, so müssen Sie nur die Position innerhalb des entsprechenden Reglers anklicken, wobei die Position am linken Rand den Wert 0 bedeutet und die Position am rechten Rand den Wert 1. Hiermit kann zwar ein genauer Wert von beispielsweise 0.765 nicht eingestellt werden, doch spielt das bei den Materialien keine allzu große Bedeutung, da es im späteren Bild nicht zu sehen ist, ob der Spiegelungsfaktor nun 0.7 oder 0.63 beträgt, es kommt also nur auf den ungefähren Wert an.

Die Farbe kann mit den ersten drei Reglern eingestellt werden, wobei je einer für Rot, Grün und Blau zu Verfügung steht. Hierbei können Sie die eingestellte Farbe in dem Rechteck, daß

sich in der rechten oberen Ecke befindet beurteilen. Mit der Schattenhelligkeit können Sie einstellen, wie hell es im Schatten des jeweiligen Körpers sein soll, hierbei bedeutet ein Wert nahe Null, daß der Schatten ziemlich dunkel sein wird, während ein Wert nahe eins einen recht hellen Schatten erzeugt. Welchen Wert man hier einstellen soll, kann nicht allgemein gesagt werden, da hier der persönliche Geschmack eine große Rolle spielt. Wollen Sie einen hohen, harten Kontrast, so muß ein Wert von ungefähr 0.1 eingestellt werden, was für meinen Geschmack ganz gut aussieht. Dies hat aber den Nachteil, daß Objekte, die sich im Schatten befinden, kaum noch sichtbar sind. Probieren Sie mit diesem Faktor am besten ein bißchen herum, genauso wie mit dem Spiegelungsfaktor.

Der gibt an, wieviel von dem Licht, das auf den Körper scheint, reflektiert werden soll. Soll der Körper matt erscheinen, so ist ein Faktor von Null nötig, soll er jedoch stark spiegeln, so muß dieser Wert natürlich größer sein. Stellen Sie hier Eins ein, so wird alles Licht reflektiert, der Körper hat dann keine eigene Farbe mehr. Wollen Sie also ein Material eingeben, so müssen Sie im Menü "MAT.EDITOR" den Punkt "Material" auswählen, oder Sie drücken am besten die Taste "m". Hiernach kann die Eingabe beginnen. Sind alle Werte richtig eingestellt, so klicken Sie "OK" an. Mit den Funktionen "links" und "rechts" können Sie wie gewohnt in der Materialliste zum Vorgänger oder Nachfolger des aktuellen Elementes gelangen. Soll ein Material gelöscht werden, so steht Ihnen hierfür der Menüpunkt "Löschen" zur Verfügung. Wollen Sie ein schon eingegebenes Material nachträglich korrigieren, so müssen Sie "Korrektur" auswählen. Daraufhin werden die Regler wieder aktiviert, und die Eingabe der gewünschten Werte geschieht wie gewohnt. Sind nun sämtliche Materialien eingestellt, so können Sie mit "quit" den Materialeditor wieder verlassen, und Sie gelangen wieder in das Hauptprogramm.

5.1.5 Die Transformationen

Unter diesem Menüpunkt finden Sie mehrere Möglichkeiten, schon eingegebene Körperdaten nachträglich zu ändern. Der einfachste Fall ist hierbei die Korrektur. Wird dieser Menüpunkt ausgewählt, so kann die gesamte Eingabe des Körpers von neuem beginnen. Mittels "Löschen" wird der Datensatz des angezeigten Körper gelöscht, und mit "Kopieren" wird er kopiert, wobei nach Ausführung des letzten Befehls die Kopie angezeigt wird. Bei den nun folgenden Funktionen können Sie nach Eingabe der eigentlichen Parameter noch einen bestimmten Modus auswählen, mit dem diese Funktion ausgeführt werden soll. Diesen Modus bestimmen Sie durch einen Tastendruck. Wird "q" gewählt, so passiert nichts. Mit "d" wird die Operation auf dem aktuellen Körper ausgeführt, und bei "c" wird der Körper vor Durchführung der Operation kopiert und dann die Transformation auf dieser Kopie angewendet. Als Transformationen stehen Ihnen zur Verfügung: die Rotation, die Vergrößerung und die Translation.

Wird die Rotation aufgerufen, so müssen die drei Winkel eingegeben werden, um die der Körper in x-, y- und z-Richtung gedreht werden soll. Hierbei bleibt der Mittelpunkt jedoch unverändert. Bei der Translation muß lediglich der neue Mittelpunkt des Körpers eingegeben werden, entweder über Tastatur oder über Maus. Ebenso einfach ist die Vergrößerung, bei der nur die drei Vergrößerungsfaktoren für die drei Hauptrichtungen angegeben werden müssen. Sind alle drei Faktoren gleich, so wird der Körper gleichmäßig in alle drei Richtungen gestreckt, andernfalls werden die Proportionen des Körpers verzerrt.

5.1.6 Die Diskettenoperationen

Um die eingegebenen Daten der Körper- und der Materialliste abspeichern und später wieder einladen zu können, stehen Ihnen im Menü "DISK" vier Operationen zur Verfügung: Loadlist, Savelist, Loadmat und Savemat. Da sich die Routinen für die Körper und die Materialien aus der Sicht des Benutzers genau

entsprechen, genügt es, die Routinen für die Körperliste zu beschreiben. Wird Loadlist aufgerufen, so verlangt das Programm den Namen des einzulesenden Datensatzes. Obwohl die Körperdaten zwar auf der Diskette mit der Endung ".list" versehen sind (bzw. die Materialien mit ".mat"), brauchen bzw. dürfen Sie diese Endung nicht mit angeben, dies wird vom Programm aus automatisch erledigt. Was jedoch nicht vom Programm aus gemacht wird, ist die Überprüfung, ob das angegebene File sich überhaupt auf der Diskette befindet. Sollten Sie also einen falschen Namen eintippen, so hält das Programm mit einer Fehlermeldung an. Ist alles richtig gelaufen, so befindet sich im Speicher nun der gewünschte Datensatz, wobei hiervon das letzte Element angezeigt wird.

Sollen umgekehrt Daten abgespeichert werden, so muß ebenfalls zu Anfang der Filename wiederum ohne Endung eingegeben werden. Zusätzlich besteht noch die Möglichkeit, nur einen bestimmten Bereich der Liste abzuspeichern, in dem der Start- und Endindex, ab bzw. bis zu dem die Daten der Liste abgespeichert werden sollen, festgelegt werden.

5.2 Bedienung des Tracers

Haben Sie alle Module zusammengesetzt und so ein lauffähiges Programm erzeugt, können Sie nun daran gehen, eigene Objekte zu editieren und zu schattieren.

Das Editieren, also das Eingeben und Ändern von Objekten, erleichtert Ihnen der eigens für diesen Zweck entwickelte Objekteditor. Da seine Funktionsweise schon beschrieben wurde, beschränken wir uns nun auf die Bedienung des Hauptprogramm-Tracers.

5.2.1 Nach dem Start

Haben Sie das Programm gestartet, können Sie, nachdem alle Librarys erröffnet wurden, die Auflösung und den Darstellungsmodus des Display-Screens bestimmen. Der Display-Screen

ist der Screen, auf dem das schattierte Bild und das Drahtmodell erscheinen.

Der Benutzer-Screen ist dagegen ist der Screen, der uns von AmigaBASIC zur Verfügung gestellt wird. Hier werden (fast) alle Eingaben von Ihnen vorgenommen. Doch zurück zur Wahl der Auflösung.

Mit den Cursortasten Hoch und Runter können Sie nun die einzelnen Zeilen, die nach dem Start auf dem Benutzer-Screen erscheinen, farbig unterlegen. In der jeweiligen farbig unterlegten Zeile können Sie dann mittels RECHTS und LINKS die Ihnen zur Verfügung gestellten Alternativen 'durchblättern'. So können Sie z.B. in der ersten Zeile den Darstellungsmodus festlegen (Normal, Hold and Modify, Extra Halfbrite), in der zweiten Zeile können Sie die Auflösung in X-Richtung (Normal und HIRES) und in der dritten Zeile die Y-Auflösung (Normal und Interlaced) festlegen.

Die letzte Zeile enthält die Anzahl der BitPlanes. Beachten Sie bitte, daß HAM und Halfbrite nur mit 6 BitPlanes in normaler X-Auflösung laufen und das Anzahl der BitPlanes sich nach dem verfügbaren Speicherplatz richtet!

Haben Sie den Darstellungsmodus und die Auflösung festgelegt, können Sie 'Return' drücken und gelangen in die Hauptschleife des Programms, in der alle Menüanfragen und Tastendrücke bearbeitet werden. Zu Beginn des Programms stehen Ihnen allerdings noch nicht alle Menüpunkte zur Verfügung. Es ist z.B. sinnlos, ein Drahtmodell zeichnen zu wollen, wenn noch gar keine Objekte im Objektspeicher enthalten sind. Erst wenn Sie Objektdefinitionen geladen oder 'gemergt' haben, können Sie alle Menüpunkte aufrufen.

5.2.2 Das 'Amiga'-Menü

Der in diesem Menü enthaltene Menüpunkt 'Info' dient, wie der Name schon sagt, zu Ihrer Information. Sie erfahren hier, wieviele Objekte maximal in dem 'Objektspeicher' untergebracht

werden können. Die Objekte (Flächen und Körper) werden im Array K() abgespeichert. Weiterhin erfahren Sie aber auch, wieviele Objekte schon im Array K() angesiedelt sind. Die Programmvariablen 'MaxAnzahl' und 'AnzahlK' enthalten diese Werte.

5.2.3 Das 'Datei'-Menü

In diesem Menü finden Sie viele Menüpunkte für die Ein- und Ausgabe auf Diskette und Drucker. Das ist aber noch nicht alles!

5.2.3.1 Laden und Speichern von Objekten

Sie können durch Wählen von 'Laden' Objektdefinitionen in den Rechner einladen. Doch bevor das zu ladende Objekt-File bestimmt werden kann, werden Sie mittels Eigenbau-Dialogboxen über das aktuelle Diskettenverzeichnis befragt.

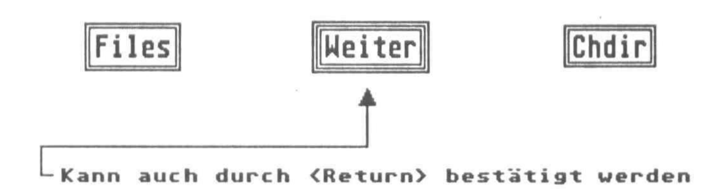


Abbildung 5.7

Dieses kann Ihnen natürlich auf Wunsch angezeigt werden ('Files'). Wollen Sie jedoch ein anderes Unterverzeichnis bestimmen, aus dem das Objekt-File geladen werden soll, so brauchen Sie nur 'Chdir' anzuklicken, und können durch Angabe eines neuen Diskettenpfades ein neues Unterverzeichnis oder eine neue Diskette auswählen. Das 'Anklicken' geschieht dabei so, daß Sie den Mauszeiger innerhalb des Rechtecks plazieren, das um einen Dialogtext gezeichnet wurde (hier 'Chdir'), und die linke Maustatste niederdrücken. Wurde ein fettes Rechteck um

einen Text gezeichnet (siehe Abbildung), so haben Sie die Möglichkeit, entweder zu klicken oder einfach 'Return' zu drücken, um auf diese Dialogbox zu reagieren.

Klicken Sie 'Weiter' an, oder drücken Sie <Return>, so gelangen Sie zur Eingabe des Filenamens. Bitte beachten Sie, daß an den eingegebenen Filenamen automatisch ".LIST" angehängt wird. Als Objektdefinitionen kommen also nur Files mit diesem 'Anhängsel' in Frage.

Konnte das angegebene File eröffnet werden (andernfalls erscheint eine Fehlermeldung), werden Sie gefragt, wieviele Objekte aus diesem File übernommen werden sollen. Vorgegeben werden Ihnen hier die gesamte Anzahl der in diesem File enthaltenen Objekte. Sie können aber auch einen größeren oder kleineren Wert eingeben. Beachten sollten Sie, daß die hier eingegebene Anzahl die maximale Anzahl der Objekte angibt, die im Objektspeicher aufgenommen werden können - oder anders gesagt: diese Zahl bestimmt die Anzahl der Elemente des Arrays K() (s. 'New'), das ja bekanntlich die Objekte enthält. Wollen Sie also später den jetzt zu ladenden Objekten noch weitere hinzufügen, so sollten Sie hier durch die Angabe einer genügend großen Zahl dafür sorgen, daß tatsächlich genügend Speicherplatz für alle Objekte vorhanden sein wird!

Dann können Sie durch die Auswahl des Menüpunktes 'Mergen' sogar noch weitere Objekte den schon vorhandenen anhängen. Auch hier werden Sie, wie übrigens bei allen Floppy-Computer-Operationen, nach dem aktuellen Unterverzeichnis (s. 'Directory') befragt.

Nachdem das File, in dem die weiteren Objekte enthalten sind, eröffnet werden konnte, werden Sie auch hier gefragt, wieviele Objekte übernommen werden sollen. Aber Achtung: wollen Sie mehr Objekte übernehmen, als im Objektspeicher Platz finden, so wird der vorhandene Speicherplatz vollkommen ausgenutzt, die überzähligen Objekte fallen aber unter den Tisch!

Sie können das vermeiden, in dem Sie in 'New' einen genügend großen Objektspeicher anlegen. Hier werden nicht wie bei 'La-

den' Objekte in den Speicher geladen, sondern eben nur der gewünschte Speicherplatz angelegt. Diesen mit 'New' angelegten Speicher sollten Sie dann mit 'Merge's auffüllen. Bei 'Laden' wird der Objektspeicher ja von Grund auf erneuert.

Und natürlich können Sie die im Speicher vorhandenen Objekte - mit 'Speichern' - auch auf Diskette schreiben. Hierbei werden aber nur die tatsächlich enthaltenen Objektjedoch jedoch auf die Diskette geschrieben. Eventuell unausgenutzte Objektspeicher (Arrayelemente des Array's K()) werden nicht abgespeichert.

Wird bei der Eingabe des Filenamens ein Leerstring, also kein Zeichen außer 'Return' eingegeben, so wird der Lade-, Speicher- oder Merge-Vorgang sofort abgebrochen und in die Hauptschleife zurückgekehrt. Es wird nicht versucht, auf die Diskette zuzugreifen.

5.2.3.2 Materialien laden

Und natürlich können Sie auch eigene Materialien, die mit dem Editor erstellt wurden, in den Rechner laden. Dabei werden Sie auch hier nach dem Unterverzeichnis der Diskette befragt. Danach brauchen Sie nur den Namen des Files anzugeben, das die neuen Materialien enthält. Diese werden dann anstatt der Default-Materialien (s. 'Initmat') beim Schattieren benutzt. Bitte beachten Sie, daß an alle Filenamen ".MAT" angehangen wird, um diese von den Objekten (".LIST") zu unterscheiden.

5.2.3.3 Hintergrund auswählen

Durch Auswahl von 'Hintergrund' haben Sie die Möglichkeit, den tristen Hintergrund des zu schattierenden Bildes ein wenig aufzuhellen. Normalerweise wird beim Schattieren nur die Hintergrundfarbe oder eine Ebene dargestellt.

Aber Sie können dies ändern: Zunächst werden Sie vor die Entscheidung gestellt, ein IFF-Bild von Diskette zu laden oder ein Muster zu erzeugen.

Entscheiden Sie sich für 'Bild laden', so erscheint erst einmal die mittlerweile schon bekannte Nachfrage nach dem Unterverzeichnis. Danach werden Sie nach dem Namen des IFF-Files gefragt, das das zu ladende Bild enthält.

Solch ein IFF-Bild kann dabei von jedem Malprogramm, das dieses Interchange-File-Format unterstützt, übernommen werden. Versuchen Sie es doch einmal mit Bildern von Graphicraft oder DeluxePaint!

Doch wieder zurück zum Hintergrund. Entscheiden Sie sich nämlich für ein Muster als Hintergrund, können Sie zwischen drei Alternativen wählen:

Muster

Klicken Sie die nun erscheinende Dialogbox 'Muster' an, so können Sie den Display-Bildschirm mit einem der Füllmuster füllen, mit deren Hilfe beim Schattieren ja die Helligkeitsabstufungen erzeugt werden. Dazu geben Sie zuerst die Nummer des gewünschten Musters ein, die zwischen 0 und 34 liegen muß (35 verschiedene Füllmuster (Standard und Extended)), dann brauchen Sie nur noch die Farbe des Vordergrund- und Hintergrundstiftes festzulegen - und schon wird der Display-Screen mit dem so bestimmten Muster in den angegebenen Farben gefüllt.

Himmel

Hier haben Sie die Möglichkeit, einen Sternenhimmel produzieren zu lassen. Auf die einzelnen Sternformen wollen wir dabei an dieser Stelle nicht weiter eingehen, da die Alternativen automatisch vom Programm aufgezeigt werden.

Nach Angabe der 'Sternart' der Farbe, in der die Sterne gezeichnet werden sollen, und der Angabe der Anzahl der zu zeichnenden 'Sterne' sehen Sie, wie diese auf dem Bildschirm 'aufgehen'.

Hier gilt wie bei den Mustern der Vorsatz: Probieren geht über studieren!

Fließender Hintergrund

Hier können Sie sich den Verlauf von einer Farbe zur anderen berechnen lassen. Dazu geben Sie einfach die Nummern der Farbregister an, zwischen deren Farben der Farbverlauf errechnet werden soll.

Befindet sich z.B. in Farbregister 1 die Farbe 'Hellgrün' und in Farbregister 3 die Farbe 'Dunkelgrün', so brauchen Sie als Farben nur '1' und '3' anzugeben, um eine Farbverlauf von Hell- nach Dunkelgrün zu veranlassen.

Generell sollten Sie sich einen Hintergrund nur dann erzeugen lassen, wenn Sie sicher sind, daß der nächste Schritt das Schattieren sein soll. Sind Sie sich z.B. noch nicht im klaren darüber, ob die aktuelle Position der Objekte richtige ist, sollten Sie erst versuchen, diese herauszufinden. Das Laden eines komprimierten IFF-Files kann nämlich zur Geduldsprobe werden.

5.2.3.4 Bilder abspeichern

Damit Sie an den mühsam errechneten Bildern auch später noch Freude haben, können Sie diese im IFF-Format auf Diskette speichern. Neben der Konservierung der Bilder hat dies auch den Vorteil, daß die schattierten Bilder mit einem Zeichenprogramm weiterbearbeitet werden können (Betextung etc.). Klikken Sie also 'Bild speichern' an, so können Sie nach Eingabe des Namens, den das File erhalten soll, das errechnete Bild auf Diskette schreiben.

5.2.3.5 Bilder ausdrucken

Konservieren können Sie Ihre Bilder allerdings auch auf Papier. 'Hardcopy' erzeugt eine Hardcopy auf den mit 'Preferences' voreingestellten Drucker. Auch ob das Bild 'Sideways' oder in 'Gray Scale' ausgegeben werden soll, bestimmen Sie mit Hilfe von 'Preferences'.

Nach der Auswahl von 'Hardcopy' erscheinen aber zunächst zwei Dialogboxen, die Sie befragen, ob tatsächlich ein Drucker angeschloßen ist. Wenn ja, so geht es nach Anklicken von 'Printer OK' sofort los. Bei 'Kein Printer' wird dieser Menüpunkt sofort verlassen und in die Hauptschleife zurückgekehrt.

5.2.3.6 Neue Farben bestimmen

Nach dem Anklicken von 'Farbpalette' werden Ihnen auf dem Display-Screen alle verfügbaren Farben gezeigt. Die Maus können Sie dann auf die zu ändernde Farbe setzten und die linke Maustaste niederdrücken, um die zu ändernde Farbe zu bestimmen (Bitte beachten Sie, daß Sie auch die Hintergrundfarbe, also Farbe 0, ändern können. Damit Sie diese aber besser erkennen können, haben wir um die Hintergrundfarbe ein kleines Rechteck gezeichnet). Nun können Sie die neue Rot-, Grün- und Blau-Komponente der Farbe bestimmen.

Danach können Sie dann entweder das 'Farbpalette' Menü verlassen ('Weiter') oder die nächste Farbe ändern.

Beachten sollten Sie noch, das beim Zeigen der verfügbaren Farben das aktuelle Bild des Display-Screens gelöscht wird! Haben Sie also vorher ein paar Objekte in stundenlanger Rechenarbeit schattiert, so sollten Sie das erzeugte Bild auf Diskette abspeichern, bevor Sie fortfahren!

5.2.3.7 Das Programm verlassen

Wenn Sie das Programm verlassen wollen, brauchen Sie nur 'Programmende' anzuwählen. Jedoch werden Sie vor dem tatsächlichen Verlassen erst noch einmal befragt, ob nicht ein Irrtum vorlag. Klicken Sie jedoch 'Ja', um das Programm wirklich zu verlassen, gibt es kein Zurück mehr! Bei 'Nein' wird das 'Programmende'-Menü verlassen, und Sie kehren in die Hauptschleife zurück.

Interressant wird es jedoch beim Anklicken von 'Neustart'. Wie der Name schon sagt, wird das Programm neu gestartet. Sie haben somit die Möglichkeit, nachträglich eine andere Auflösung zu wählen, falls Ihnen die vorhandene nicht mehr gefällt oder ausreicht. Aber Vorsicht! Auch hier lösen sich alle vorherigen Arbeiten in Rauch und Asche auf!

5.2.4 Das 'Editor'-Menü

Der einzige in diesem Menü enthaltene Menüpunkt ist 'Editor aufrufen'. Der Name spricht wohl für sich. Doch bevor der Editor aufgerufen wird, müssen Sie Ihren Wunsch noch einmal bestätigen ('Aufrufen'). Fällt Ihnen nämlich z.B. ein, daß ein vorher schattiertes Bild noch nicht auf Diskette abgespeichert wurde, so können Sie durch Anklicken von 'nicht Aufrufen' dafür sorgen, daß Sie das 'Editor aufrufen'-Menü verlassen und das Bild doch noch abspeichern können.

5.2.5 Das 'Parameter'-Menü

In diesem Menü sind im großen und ganzen die Routinen enthalten, die die Parameter für die dreidimensionale Darstellung verändern.

Mit 'Projektionspunkt' können Sie die Koordinaten des Projektionspunktes neu festlegen. Ebenso geschieht die Festlegung des Hauptpunktes in 'Hauptpunkt'. Die Drehwinkel um

die drei Koordinatenachsen können Sie in 'a,ß,c' ändern. Und den Abstand vom Projektions- zum Hauptpunkt können Sie in 'Abstand' neu eingeben.

In welchem kausalen Zusammenhang der Haupt- und Projektionspunkt sowie die Drehwinkel und der Abstand DPH aber liegen, können Sie aus dem Kapitel 'Von 3-D nach 2-D' erfahren.

Auch 'Anzahl Ecken' wurde schon erläutert. Sie erinnern sich: die hier geänderte Zahl (AnzahlSegmente) gibt die Anzahl der Ecken eines Kreises an.

Während in den Routinen der oben genannten Menüpunkte nur Zahlen eingegeben werden mußten und keinerlei Dialogboxen Entscheidungen von Ihnen abverlangten, ist dies bei 'Vergrößern' ganz anders.

Wie Sie ja wissen, werden die Körper bzw. Objekte als Alternative zum sehr langsamen Schattieren als Drahtmodell gezeigt. Vergrößern Sie die Körper des Drahtmodells, so werden analog die Körper beim Schattieren auch größer dargestellt.

Sie haben nun aber zwei Möglichkeiten des Vergrößerns:

Proportional

Hier geben Sie einen Vergrößerungsfaktor vor. Das Bild wird dann in X- und in Y-Richtung gleichermaßen vergrößert.

Verzerrt

Beim verzerrten Vergrößern muß dagegen von Ihnen ein Bildschirmausschnitt mit der Maus festgelegt werden, der dann den gesamten Bildschirm ausfüllen soll. Drücken Sie den linken Mausknopf nieder, so wird an der aktuellen Mausposition die linke obere Ecke eines Rechtecks festgelegt. Bewegen Sie die

Maus mit niedergedrücktem Knopf hin und her, so können Sie sehen, wie ein sich analog zur Mausbewegung vergrößerndes und verkleinerndes Rechteck gezeichnet wird.

Haben Sie das richtige Rechteck bestimmt, lassen Sie den Mausknopf wieder los. Sie werden dann gefragt, ob der gewählte 'Ausschnitt OK' ist. Wenn ja, dann klicken Sie bitte diese Dialogbox an. Wollen Sie aber einen anderen Ausschnitt vergrößern, so klicken Sie einfach an irgendeiner Stelle außerhalb der 'OK'-Box, und schon geht die Ausschnittwahl von neuem los.

War der Ausschnitt aber 'OK', wird der 'Inhalt' des so bestimmten Rechtecks vergrößert.

Allerdings müssen Sie diesen Ausschnit bei einer Proportionalvergrößerung mit dem Vergrößerungsfaktor '1' auswählen. Haben Sie z.B. vorher eine Proportionalvergrößerung von '4' gewählt, wollen dann aber einen Ausschnitt vergrößern, wird nicht das vergrößert, was Sie gerade im Ausschnittrechteck eingefangen haben, sondern das, was sich bei einer X-,Y-Vergrößerung von 1 an dieser Stelle befindet!

So kann es unter Umständen passieren, daß Sie nach dem Vergrößern gar nichts mehr auf dem Display-Screen sehen. Dem kann aber abgeholfen werden, indem Sie nochmals 'Vergrößern' anwählen, diesmal aber proportional vergrößern, einen Faktor von '1' angeben und dann nochmals verzerrt vergrößern.

5.2.6 Das 'Zeichne'-Menü

In diesem Menü finden Sie Unterpunkte sowohl für das Schattieren als auch für das Zeichnen des Drahtmodells.

5.2.6.1 Schattieren

Nun kommen wir endlich zum Schattieren. Wählen Sie den Menüpunkt 'Schattieren', wird mit dem Schattieren begonnen.

Meist wollen Sie aber vorher bestimmte Parameter für das Schattieren festlegen (Schattierfenster etc.).

Dies geschieht nach Anwahl von 'initialisieren'. Hier werden Sie zunächst nach dem Schattierfenster befragt. Klicken Sie die Dialogbox 'Alles' an, wird der gesamte Bildschirm schattiert. Bei 'Ausschnitt' haben Sie die Möglichkeit, einen Bildschirmausschnitt zu wählen.

'YA-YE' bietet Ihnen die Möglichkeit, einen vertikalen Bereich, also einen Streifen, zum Schattieren auszuwählen. Dieser Streifen hat dabei die volle Breite des Bildschirms, Sie können seine Position und Höhe bestimmen.

Dazu bewegen Sie die Maus zuerst auf die Anfangsposition der zu schattierenden Leiste. Eine Gerade, die immer an der aktuellen Mausposition gezeichnet wird, hilft Ihnen bei der Positionierung. Haben Sie die richtige Position gefunden, brauchen Sie nur den linken Mausknopf niederzudrücken und die Endposition anzuvisieren. Danach lassen Sie den Mausknopf einfach wieder los - und der Schattierstreifen ist ausgewählt.

Sie können sich aber auch entscheiden, zuerst die End- und dann die Anfangsposition der Leiste zu bestimmenn. Die Reihenfolge ist egal. Das Programm sorgt dafür, daß die größere Y-Koordinate die End- und die kleinere Y-Koordinate die Anfangsposition bestimmen.

Doch kommen wir nun zu den weiteren Parametern für das Schattieren. Sie können jetzt nämlich darangehen, die Lichtquelle im Raum zu plazieren. Dazu geben Sie einfach die X-, Y- bzw. Z-Koordinate der Lichtquelle in 'Qx','Qy' und 'Qz' an. Bitte beachten Sie, daß die Lichtquelle auch um die Koordinatenachsen gedreht wird!

Auch die Farbe der Lichtquelle können Sie bestimmen. Dazu geben Sie einfach die Rot-, Grün- und Blau-Komponente der Lichtquelle in Werten zwischen 0 und 1 an - genauso wie bei der PALETTE-Anweisung.

Beachten Sie bei der Eingabe von Werten zwischen 0 und 1, daß die 0 vor dem Komma unbedingt angegeben werden muß. '.5' anstatt '0.5' ist also nicht erlaubt.

Weil das Schattieren unter Umständen mehrere Tage in Anspruch nehmen kann, haben wir die Größe der zu setzenden Punkte variabel gestaltet. Dies hat den Vorteil, daß Sie sich zuerst ein grobes Bild innerhalb von wenigen Stunden errechnen lassen können. Gefällt Ihnen das Ergebnis, können Sie für einen erneuten Schattierdurchgang die 'Punkthöhe' und 'Punktbreite' auf 1 setzen und somit alle Punkte des Schattierfensters schattieren. Haben Sie an dem ungefähren Ergebnis noch etwas auszusetzen, können Sie diese 'Fehler' durch eine Änderung der Parameter beseitigen.

Wählen Sie 'Punkthöhe' und 'Punktbreite' aber größer als 1, wird eben nur jeder zweite, dritte, vierte,... Punkt zur Farbberrechnung herangezogen. Das Bild wird also in einem 'Punktbreite'*'Punkthöhe'-Raster schattiert. Der Mittelpunkt eines solchen Rasterrechtecks bestimmt dabei die Farbe des gesamten Rechtecks.

Nachdem alle diese Parameter von Ihnen eingegeben worden sind, können Sie nun den Menüpunkt 'Schattieren' anwählen.

Sie können aber auch die Defaultwerte fürs Schattieren übernehmen. Hierbei wird der gesamte Bildschirm schattiert und die Lichtquelle mit der Farbe (1,1,1) = Weiß an die Koordinate (1000,1000,1000) gesetzt. 'Punkthöhe' und 'Punktbreite' sind gleich 1.

Während des Schattierens ist Ihr Computer natürlich nicht ansprechbar. Drücken Sie aber während des Schattiervorgangs eine beliebige Taste nieder, so werden Sie, nachdem die aktuelle Zeile zu Ende schattiert wurde, befragt, ob das Schattieren 'Weiter' gehen, oder ge'Stop't werden soll.

So können Sie z.B. den Computer nachts schattieren lassen, morgens den aktuellen 'Schattierstand' abspeichern und andere Aufgaben mit Ihrem Amiga erledigen.

Natürlich könnte man einwenden, daß der Amiga multitaskingfähig ist. Deshalb sollten Sie AmigaBASIC auch über das CLI starten, um über das Amiga-DOS-Window z.B. noch Disketten formatieren zu können.

Aber ein ernsthaftes Arbeiten im Multitaskingbetrieb während des Schattierens ist nicht zu empfehlen. Stürzt nämlich irgendein anderer Task ab - und das geschieht vielen Programmen noch recht häufig - und erzeugt eine 'Guru-Meditation', ist auch Ihr schattiertes Bild futsch!

5.2.6.2 Das Drahtmodell

Wählen Sie 'Drahtmodell', so wird das Drahtmodell der Objekte entweder neu gezeichnet, oder es bleibt einfach bei der Umschaltung von Benutzer- auf den Display-Screen.

Dies ist dann der Fall, wenn kein Parameter (Projektionspunkt, Hauptpunkt, Abstand zur Projektionsfläche, Drehwinkel) nach dem letzten 'Zeichneneu' verändert wurde. Wird aber einer dieser Parameter zwischenzeitlich verändert oder durch 'Bild löschen' der Display-Screen gelöscht oder ein Hintergrund ausgewählt, wird das Bild neu gezeichnet.

Allerdings wird nicht wie sonst nach der Auswahl eines Hintergrundes der Display-Screen vor dem Neuzeichnen gelöscht - sonst wäre die Hintergrundauswahl ja umsonst gewesen. Das Drahtmodell wird über den Hintergrund gezeichnet.

Ist das Drahtmodell fertig gezeichnet, blitzt der Bildschirm kurz auf. Das Drahtmodell bleibt dann solange auf Ihrem Bildschirm, bis Sie irgendeine Taste drücken. Danach gelangen Sie wieder in die Hauptschleife zurück.

5.2.7 Steuerung des Programms über Tastatur

Um denjenigen unter Ihnen, die die Tastatur als Eingabemittel bevorzugen, ein wenig näher zu kommen, haben wir verschiedene Menüpunkte auch durch einfache Tastendrücke (Shortcuts) erreichbar gemacht. So können Sie z.B. durch <SPACE> für ein 'Drahtmodell' sorgen.

Die Menüpunkte, die auch über Tastatur angesprochen werden können, enthalten die zu drückende Taste in ihrer Menüzeile. Es gibt aber auch Funktionen, die nicht über das Menü, sondern über Tastatur aufzurufen sind.

Eine davon ist 'Help'. Durch Drücken der Help-Taste erscheint auf dem Benutzer-Screen eine Liste der gängigen Tastaturcodes und der Aktionen, die durch sie ausgelöst werden.

So erfahren Sie z.B. daß die Drehwinkel mittels der Cursortasten und der Taste 'R' und <CTRL R> erhöht und vermindert werden können. Und auch der Abstand zur Projektionsfläche (DPH) kann mittels '+ - * /' verändert werden.

Das war's im wesentlichen zur Tastatur- und Menüsteuerung des Programms. Noch tiefergehende Erläuterungen wären ziemlich zwecklos, da Sie nur durch häufige Benutzung zu 'Tracer-Profis' werden können. Also, viel Spaß!

6. Ausblicke und Erweiterungen

Natürlich stellen die in diesem Buch beschriebenen Algorithmen und die Routinen auf der Diskette noch nicht der Weisheit letzten Schluß dar. Hier und da sind immer kleinere oder größere Verbesserungen möglich. In diesem Kapitel soll ein Teil dieser Möglichkeiten aufgezeigt werden, damit Sie das Programm nach und nach perfektionieren können.

Fangen wir mit den Grundobjekten an. Bei den Grundobjekten, die wir bereits definiert haben, können wir sicherlich noch mehr Einschränkungen angeben als die, die wir schon besprochen haben. Möglich sind zum Beispiel Kegelausschnitt, Kegelstumpf, Ellipsoidausschnitt, Ellipsoidabschnitt etc. Da sich deren Einschränkungen nicht wesentlich von den bereits bekannten unterscheiden, dürfte es Ihnen keine Probleme machen, die entsprechenden Unterprogramme, vor allem für die Schnittpunktsberechnung, aus den vorhandenen zusammenzustellen.

Ebenso sind auch neue Grundobjekte denkbar. Überlegen Sie sich doch einmal, was passiert, wenn man die Einschränkung des Ellipsoids zu u^10+v^10+w^10=1 ändert. Es wäre doch interessant auszuprobieren, was dabei herauskommt. Vielleicht ein Würfel mit abgerundeten Ecken? Ändern Sie auch mal die Einschränkungen der anderen Grundobjekte ab, vielleicht indem Sie statt des Quadrates die Wurzel berechnen oder ähnliches.

Als ganz neues Grundobjekt böte sich der Torus an. Das Problem dabei: Es gibt maximal vier Schnittpunkte. Versuchen Sie doch mal, dessen Gleichung aufzustellen und daraus dann das Unterprogramm SchnittpunktTorus zu machen. Knifflig wird es spätestens bei der Berechnung des Normalenvektors, aber lassen Sie sich dadurch nicht abschrecken. Falls Sie es nicht schaffen, trösten Sie sich damit, daß ich es auch (noch) nicht geschafft habe.

Oder lassen Sie sich ein völlig anderes Grundobjekt einfallen. Blättern Sie Ihre Formelsammlung durch, vielleicht finden Sie ein paar interessante Dinge, die Sie verwenden können, um das Programm weiter aufzuwerten.

Eine andere Möglichkeit, neue Formen zu erzeugen, besteht darin, diese aus vielen Dreiecken zusammenzusetzen. Leider ist es ziemlich mühselig, jedes einzelne Dreieck einzugeben, ganz zu schweigen vom Berechnen von Position und Größe. Hier gibt es die Möglichkeit, über Bezierfunktionen oder B-Splines nahezu beliebige Oberflächen durch Angabe weniger Punkte erzeugen zu lassen.

Beide Verfahren versuchen, eine Oberfläche zu erzeugen, die keine Ecken und Kanten hat, sondern nur "weiche Übergänge". Dabei verläuft bei B-Splines die Oberfläche durch die angegebenen Punkte, während bei Bezier die Punkte außerhalb der Oberfläche liegen. Das problematische an diesen beiden Verfahren ist, daß eine Oberfläche schon in sehr viele Dreiecke zerteilt werden muß, damit diese hinterher wirklich glatt wirkt (Wieviel MB RAM haben Sie eigentlich?).

Auf der anderen Seite steht aber der große Vorteil, daß man mit wenig Aufwand wunderbar geschwungene Formen hinbekommt. So läßt sich ein Weinglas, das der Computer berechnen soll, bereits durch ein paar dutzend Punkte definieren. Mittels Bezier oder B-Spline werden daraus ein paar tausend Dreiecke erzeugt, die dann eine nahezu perfekte Illusion erzeugen.

Zusätzlich zu Bezier oder B-Spline kann man noch Smooth-Shading einsetzen. Smooth-Shading verwischt die Kanten von aneinander stoßenden Flächen, indem es den Normalenvektor für jede der beiden Flächen so berechnet, daß sich dieser kontinuierlich ändert und nicht sprunghaft an der Stoßstelle. Dazu müssen Sie für jede Kante der Oberfläche definieren, welche andere Oberfläche daran stößt. Die Berechnung ist dann nicht mehr übermäßig schwierig. Sie müssen zusätzlich noch den Normalenvektor der anderen Oberfläche ermitteln und beide in Abhängigkeit des Abstandes des Schnittpunktes von der Stoßkante verknüpfen. Befindet sich der Schnittpunkt sehr weit von der Stoßkante weg, so nehmen Sie den ursprünglichen Normalenvektor, befindet sich der Schnittpunkt direkt an der Stoßkante, so addieren Sie die Normalenvektoren beider Flächen im Verhältnis 50 zu 50.

Durch Smooth-Shading brauchen Sie bei Bezier und B-Spline nicht soviele Dreiecke ausrechnen zu lassen, da die Kanten ja geglättet werden. Allerdings ist der Rechenaufwand größer, so daß auch die Rechenzeit steigt. Nachteil von Smooth-Shading ist, daß nur die Normalenvektoren gesmoothed werden. Der Umriß der Oberfäche ändert sich nicht. So können Sie die Oberfläche eines Würfel durch Smooth-Shading zu der einer Kugel machen, der Umriß bleibt allerdings nach wie vor ein Würfel.

Eine weitere Vereinfachung bei der Eingabe von Objekten stellen Rotationskörper dar. Diese können aus Zylindern, Kegeln und Kreisflächen zusammengesetzt werden. Sie müssen nur einen kleinen Editor schreiben, mit dem Sie den Schnitt eines Rotationskörpers eingeben können, und welcher daraus dann die entsprechenden Daten für die einzelnen Objekte erzeugt und auf Diskette abspeichert. Sie können sogar einen Ausschnitt aus einem Rotationskörper erzeugen lassen, müssen dazu aber auch den Kegelausschnitt implementiert haben.

Wie Sie sehen, bieten bereits die Grundkörper genug Spielraum für eigene Verbesserungen. Weitere Gestaltungsmöglichkeiten haben Sie, indem Sie die Oberflächenstrukturen der Objekte verändern.

Bis jetzt hatten alle unsere Oberflächen gemeinsam, daß sie glatt waren. Zwar konnten wir verschiedene Materialkonstanten definieren, diese beeinflußten aber nur die Farbe und den Spiegelungskoeffizienten der Oberfläche. Wie wär's, wenn wir für Oberflächen auch ein beliebiges Relief definieren könnten? Auch dieses Problem läßt sich rein rechnerisch lösen. In Abhängigkeit von einer Funktion, welche die Art des Reliefs bestimmt, wird der Normalenvektor "verbogen". Passiert dies in Abhängigkeit von der Position und wiederholt es sich periodisch, so haben wir ein regelmäßiges Relief. Ein, wenn auch kleiner, Nachteil des Reliefs ist, daß es wieder nur den Normalenvektor betrifft; die Oberfläche selbst bleibt glatt. So sollten Sie stets kleine Reliefs verwenden, da dies bei diesen nicht auffällt.

Bei Ebenen definieren wir die Relief-Funktion in Abhängigkeit von den beiden Parametern der Richtungsvektoren u und v, bei Kugeln und Ellipsoiden nehmen wir die Winkel der Polarkoordinaten des Schnittpunkts, und bei Zylinder und Kegel nehmen wir den Polarkoordinatenwinkel, der für die Drehung um die r4-Achse zuständig ist, und den Parameter für die r4-Achse, also w. Das Erstellen der Relief-Funktion überlasse ich Ihnen und Ihrer Phantasie.

Für weitere Abwechslung im Einerlei der Oberflächen sorgen Muster. Ob es nun Schachbrettmuster sind oder ob die Oberfläche gepunktet, gestreift oder mit kleinen Herzchen versehen ist, bleibt auch hier ganz Ihnen überlassen.

Muster ändern nicht den Normalenvektor einer Oberfläche, sondern deren Farbe oder gleich deren gesamte Materialkonstanten. Auch hier sollte das Muster periodisch und in Abhängigkeit von den Parametern oder den Polarkoordinaten definiert sein. Am besten ist es, wenn Sie das Muster im Feld Mat definieren. Nehmen Sie dazu beispielsweise Mat(n,6), welches ja noch nicht belegt ist. Wenn dieses Element Null ist, so gelten alle anderen Elemente, wie wir es definiert haben. Andernfalls gibt Mat(n,6) die Nummer des Musters an, also zum Beispiel:

```
1: Schachbrett.
2: Gepunktet.
3: Liniert.
4: ...
```

Mat(n,0) bis Mat(n,5) enthalten dann die Materialnummern für die verschiedenen Materialien des Musters. Beim Schachbrett könnten wir den schwarzen Felder andere Materialkonstanten zuordnen als den weißen. Die oben erwähnten Herzchen könnten mehrfarbig sein oder Sie könnten ein rot-blaues Commodore-Symbol definieren. Apropos Definieren: Definieren können wir Muster ebenso wie Reliefs wieder über die Parameter und die Polarkoordinaten. Natürlich können Sie sich auch andere Methoden einfallen lassen, zum Beispiel in Abhängigkeit von der Z-Koordinate, so daß Sie Höhenlinien erhalten.

Und wo wir gerade bei Oberflächen und deren Gestaltung sind: Wie sähe es wohl aus, wenn wir ein Bild auf eine Oberfläche projizieren würden. Zwar reicht dazu unsere momentane Datenstruktur nicht aus, aber die können wir ja erweitern. Wir müßten dann eine Möglichkeit in das Programm einbauen, um ein Bild im IFF-Format zu laden. Ferner müßten wir spezifizieren können, wo das Bild auf der Oberfläche liegen soll und ob es mehrfach nebeneinander liegen soll, also so wie unsere Muster.

Statt der Farbe, die im Mat-Feld für die Oberfläche definiert ist, nehmen wir dann die Farbe, die im Bild verwendet wurde. X- und Y-Koordinate des Bildes errechnen wir in bewährter Weise aus den Parametern/Polarkoordinaten. Die anderen Materialeigenschaften, wie Spiegelung etc., könnten erhalten bleiben. Stellen Sie sich eine Flasche vor, die Sie mit Bezier definiert, mit Smooth-Shading gesmoothed und mit einem Ettiket verziert haben, welches Sie mit einem Malprogramm designed haben. Allerdings treiben wir unseren Amiga damit an die Grenzen seiner Fähigkeiten, nicht nur von der Rechenzeit, sondern auch von der Auflösung und der Farbpracht her.

Deshalb wollen wir uns nun anderen Dingen zuwenden, welche durchaus in einem realistischen Rahmen liegen, zum Beispiel der Transparenz. Wie wir Transparenz simulieren, haben wir ja schon besprochen. Allerings reicht es nicht aus, einfach noch einen BerechnePunkt-Aufruf in das entsprechende Unterprogramm einzubauen. Vielmehr müssen wir auch berücksichtigen, daß transparente Objekte Licht passieren lassen, so daß unter einem transparenten Objekt kein Schatten ist, sondern Licht, dessen Farbe allerdings nicht mehr unbedingt mit der ursprünglichen Farbe übereinstimmt.

Einbauen können wir Transparenz, für die wir glücklicherweise bereits Platz in unserer Materialdatenstruktur gelassen haben, indem wir in BerechnePunkt, vor dem rekursiven Aufruf von BerechnePunkt für die Spiegelung, einen weiteren Aufruf dieser Routine einbauen. Allerdings bleibt dabei der Richtungsvektor des Sehstrahls (rx,ry,rz) unverändert. Wir müssen darauf achten, daß sich alle Variablen, die danach für den zweiten Berechne-

Punkt-Aufruf gebraucht werden, nicht verändern, also gegebenenfalls gesichert werden müssen. Zusätzlich muß in WelcherKoerper, falls Schatten! True ist, zuerst das Objekt gesucht werden, das Schatten auf den fraglichen Punkt wirft, und das der Lichtquelle am nächsten ist. Dann muß die Farbe des Lichtstrahls entsprechend der Materialstruktur dieses Objekts geändert werden. Danach muß das Objekt gesucht werden, der Lichtquelle am zweitnächsten ist und so weiter, bis alle Objekte, die Schatten auf den fraglichen Punkt werfen, bekannt sind. Dann muß bei der Berechnung von Hell.r, Hell.g und Hell.b die errechnete Farbe berücksichtig werden. Die Berücksichtigung der Farbe der Lichtquelle, die momentan erst vor dem Zeichnen des Punktes auf den Bildschirm erfolgt, muß dann entsprechend vorgezogen werden, darf aber im Falle von Schatten nicht doppelt ausgeführt werden.

So einfach, wie sich die Spiegelung beherrschen ließ, ist es bei der Transparenz nicht mehr; und beide zusammen machen die Sache noch komplizierter. Bei der Definition der Materialien sollte die Summe aus Spiegelungs- und Transparenzkoeffizient nicht größer als Eins sein, da die Helligkeit der Oberfläche sonst eventuell ebenfalls größer als Eins werden könnte, was wiederum einem realistischen Eindruck abträglich wäre.

Als letztes Feature fällt mir noch die Möglichkeit ein, mehrere Lichtquellen definieren zu können. Wir müßten dann alle Berechnungen für jede Lichtquelle einmal durchführen und die Farben, die wir erhalten, addieren. Auch hier sollte die Addition der Helligkeiten aller Lichtquellen keine Werte über Eins ergeben, da unser Bild sonst leicht überbelichtet werden kann. Um auch die Optimierungen nutzen zu können, muß für jede Lichtquelle ein MinMaxLq-Feld berechnet werden, oder wir erweitern das MinMaxLq-Feld um eine Dimension, welche dann die Nummer der Lichtquelle angibt:

```
DIM MinMaxLq(AnzahlLQ,AnzahlK,6)
```

Vielleicht fallen Ihnen ja auch mehr Optimierungen ein, nicht nur für mehrere Lichtquellen. Oder Sie haben Ideen für weitere Gestaltungsmöglichkeiten. In diesem Kapitel sollten nur Anstöße gegeben werden, die Ihnen die Möglichkeiten zeigen sollen, die im Ray-Tracing stecken. Diese Möglichkeiten zu nutzen und weitere zu finden, bleibt Ihnen überlassen. Zwar werden auch wir, die Autoren, weiter an unseren Algorithmen herumfeilen, aber vielleicht haben Sie ja viel bessere Ideen als wir. Computergrafik ist eine fazinierende Angelegenheit, machen wir das beste daraus!

7. Mathematische Grundlagen

Haben Sie Angst vor Mathematik? Wirklich? Nun, vielleicht liegt das daran, daß Sie immer das Gefühl hatten, daß Mathematik irgendetwas Geniales ist, das nur ausgesprochene Könner beherrschen. Dabei kann Mathematik so (schön) einfach sein. Im weiteren Verlauf dieses Kapitels möchten wir Ihnen einige Grundlagen der Mathematik in wenigen Sätzen näherbringen. Falls Sie noch Fragen haben, verweisen wir Sie an mathematische Fachliteratur, da ansonsten der Rahmen dieses Buches gesprengt würde.

7.1 Vektorrechnung

Zuerst wollen wir uns den Vektoren widmen. Ein Vektor ist ein Repräsentant einer (Pfeil-) Klasse. Solch ein Pfeil hat einen Anfangspunkt und eine Pfeilspitze, die auf den Endpunkt zeigt.



Abbildung 7.1

Vektoren mit gleicher Länge, Richtung und Orientierung sind dabei gleich. Die Richtung wird durch die Pfeilspitze angegeben, und die Orientierung durch die Lage. Dabei nennt man auch Vektoren 'gleich', die nicht am gleichen Punkt beginnen, sich aber trotzdem ähneln.

In der Geometrie unterscheidet man grundsätzlich zwei Arten von Vektoren:

1. Ortsvektoren und
2. Richtungs- oder Differenzvektoren

Ortsvektoren 'starten' dabei immer im Ursprung des Koordinatensystems, also vom Punkt (0,0,0) aus. Ortsvektoren legt man meist so fest, daß man nur die Endpunkt angibt - der Anfangspunkt ist ja fest bestimmt. So kann man also auch Punkte des Raumes als (Orts-)Vektoren auffassen.

Richtungs- oder Differenzvektoren starten jedoch nicht vom Ursprung aus und sind - wie der Name schon sagt - meist der Differenzvektor zweier (Orts-)Vektoren. Was Richtungsvektoren zu Richtungsvektoren macht, ist die Tatsache, daß der Startpunkt meist der Endpunkt eines Ortsvektors ist.

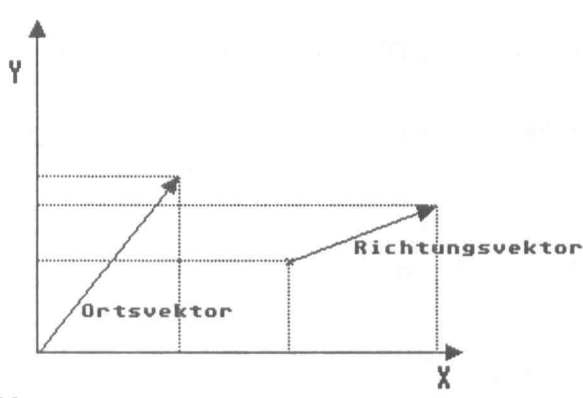


Abbildung 7.2

Doch durch die Nennung der Differenzvektoren haben wir schon ein wichtiges Thema angesprochen: Das Rechnen mit Vektoren. Man kann Vektoren nämlich addieren und subtrahieren:

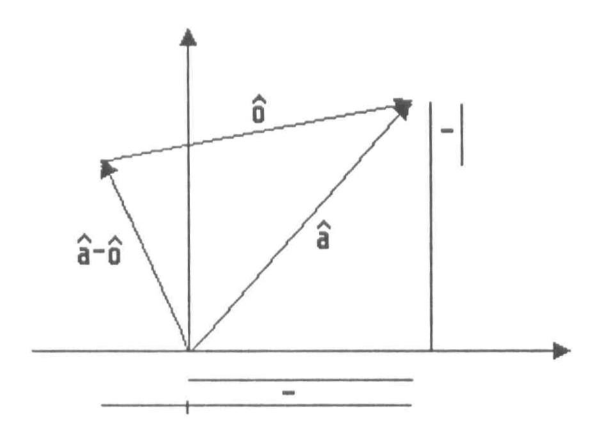


Abbildung 7.3

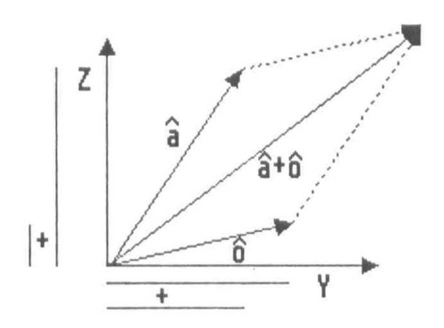


Abbildung 7.4

Dabei addiert oder subtrahiert man ganz einfach die Komponenten der Vektoren (Man spricht von X,Y,Z-Komponente eines Vektors, und von X,Y,Z-Koordinate eines Punktes!):

$$\hat{a}\pm\hat{o} = \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} \pm \begin{pmatrix} ox \\ oy \\ oz \end{pmatrix} = \begin{pmatrix} ax\pm ox \\ ay\pm oy \\ az\pm oz \end{pmatrix} = \hat{u}$$

Bei der Multiplikation eines Vektors mit einem Skalar (einer reellen Zahl) multipliziert man jede Komponente mit dieser Zahl:

$$t * \hat{a} = t * \begin{pmatrix} ax \\ ay \\ az \end{pmatrix} = \begin{pmatrix} t*ax \\ t*ay \\ t*az \end{pmatrix} = \hat{u}$$

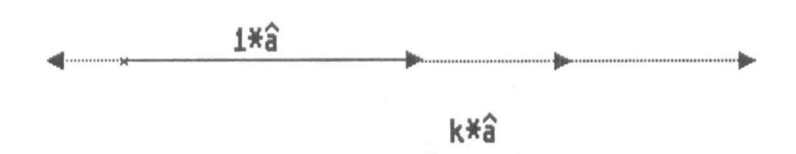


Abbildung 7.5

7.2 Was man mit Vektoren machen kann

Was kann man aber mit Vektoren anstellen? Nun, man kann eigentlich alle Körper und Flächen durch (mehr oder weniger viele) Vektoren beschreiben.

Aber auch Geraden lassen sich mittels Vektoren darstellen. Dazu nimmt man einen Ortsvektor, der einen Punkt der Geraden festlegt, und einen Richtungsvektor, der an den Ortsvektor 'angehangen' wird. Mit folgender Geradengleichung kann man eine Gerade beschreiben:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \hat{o}x \\ \hat{o}y \\ \hat{o}z \end{pmatrix} + l * \begin{pmatrix} rx \\ ry \\ rz \end{pmatrix}$$

Grafisch sieht das Ganze so aus:

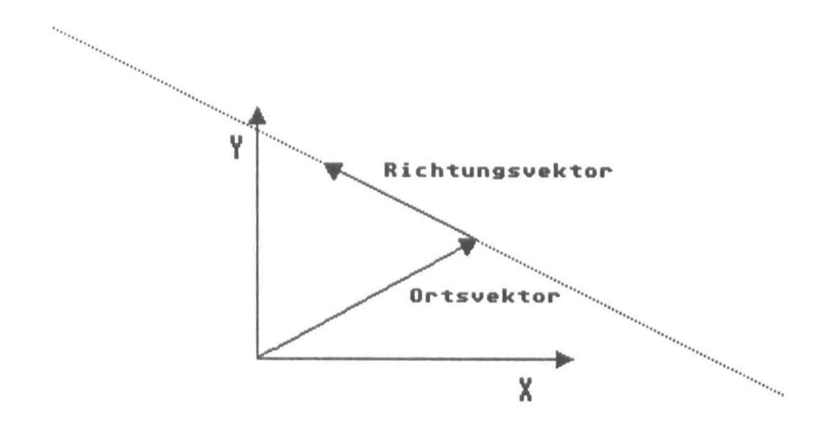


Abbildung 7.6

Der Richtungsvektor wird also wie ein Teleskoparm in Abhängigkeit von 'l' ausgefahren und greift so alle Punkte der Geraden ab.

Ebenen werden auf ähnliche Art und Weise dargestellt. Nur werden hier zwei Richtungsvektoren benötigt, die linear unabhängig sein müssen. Linear unabhängig bedeutet dabei nur, daß der eine Vektor nicht ein vielfaches (S-Multiplikation) des anderen Vektors ist.

Die Ebenengleichung sieht so aus:

```
x     ôx     r1x     r2x
y  =  ôy + a*r1y + b*r2y
z     ôz     r1z     r2z
```

Etwas schwieriger zu verstehen ist die Kugelgleichung:

```
( x     Mx)²
( y  -  My)² = r²
( z     Mz)²
```

Doch diese Abbildung macht's wohl etwas deutlicher:

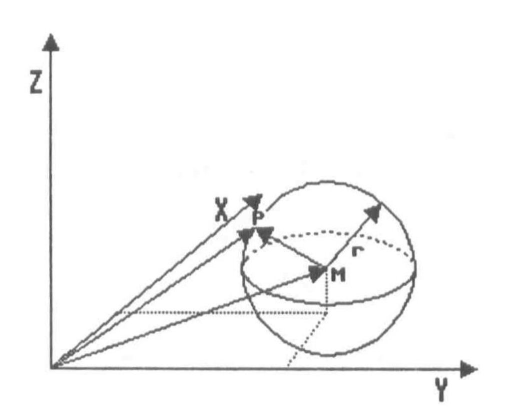


Abbildung 7.7

Befremdlich an der Kugelgleichung ist wohl, daß der Punkt auf der Oberfläche der Kugel in die Gleichung 'eingebaut' wurde. Und auch die Quadrierung der Vektoren ist neu. Aber wenn man sich die Differenz der beiden Vektoren ansieht (Punkt auf der Kugel - Mittelpunkt), wird klar, daß dieser Differenzvektor die gleiche Länge haben muß wie der Radius, falls der Punkt auf der Oberfläche der Kugel liegt.

Jetzt könnte man sich dennoch fragen, warum quadriert wird. Dies liegt daran, daß man einen Vektor nicht so ohne weiteres mit einem Skalar (einer reellen Zahl) vergleichen kann. Durch die Quadrierung wird aber durch das sogenannte Skalarprodukt eine reelle Zahl bestimmt:

$$\begin{pmatrix} x^2 \\ y^2 \\ z^2 \end{pmatrix} = x*x+y*y+z*z = a$$

Ist diese Zahl gleich dem Quadrat des Radius, so weiß man:

Der Punkt (Ortsvektor) $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ liegt auf der Kugel.

Dies waren also die Grundkörper und Flächen, die man mit Vektoren darstellen kann.

7.3 Der Winkel zwischen zwei Vektoren

Natürlich kann man auch den eingeschlossenen Winkel zwischen zwei Vektoren bestimmen. Dazu schweifen wir aus dem Raum in die Ebene ab und betrachten zwei senkrecht aufeinanderstehende Vektoren:

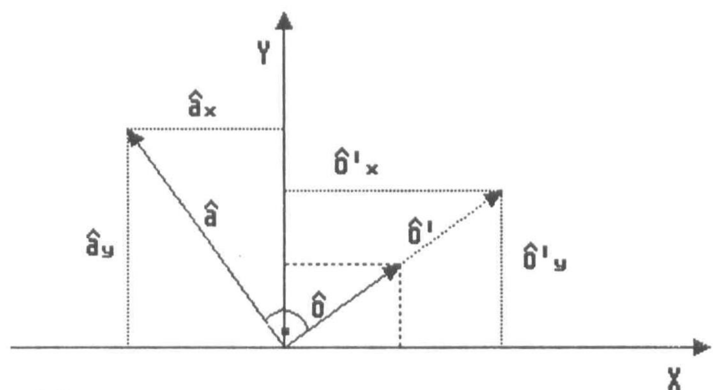


Abbildung 7.8

Den einen Vektor (ô) kann man durch S-Multiplikation so weit 'ausfahren', daß er genauso lang ist wie der andere (â). Betrachtet man die Koordinaten bzw. Komponenten der Vektoren, dann stellt man fest, daß gilt:

$$ax = -o'y \qquad ay = o'x$$

Da ô' aber ein vielfaches des Vektors ô ist, kann man für ô' auck k*ô schreiben:

$$ax = -k*oy \qquad ay = k*ox$$

Durch Auflösen nach 'k', anschließendem Einsetzen und Umformen erhält man:

```
ax*ox+ay*oy = 0
```

Dies ist auch eine Form des Skalarproduktes (s. o.), hier allerdings auf zwei Vektoren bezogen.

Betrachtet man nun zwei nicht senkrecht aufeinanderstehende Vektoren, dann gilt:

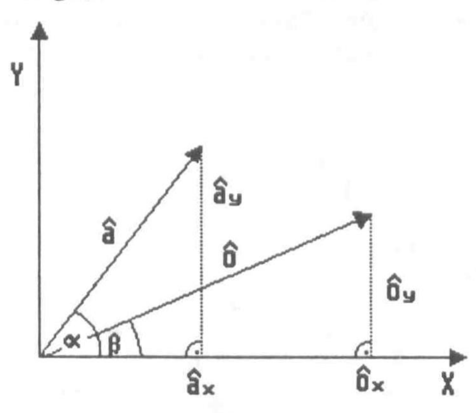


Abbildung 7.9

```
ax = Cos(α) * |â|          ox = Cos(ß) * |ô|
ay = Sin(α) * |â|          oy = Sin(ß) * |ô|
```

Setzt man dies in die Gleichung des gerade gefundenen Skalarprodukts ein, gilt weiterhin:

```
  Cos(α) * |â| * Cos(ß) * |ô|
- Sin(α) * |â| * Sin(ß) * |ô| = ax*ox +ay*oy

<=> (Cos(α)*Cos(ß) - Sin(α)*Sin(ß)) * |â| * |ô| = ax*ox+ay*oy
```

Den Ausdruck mit den vielen Cosinus und Sinus kann man aber mittels der Additionstheoreme des Cosinus (Cos (ß-α) = (Cos α*Cos ß + Sin α*Sinß)) noch vereinfachen:

$$\text{Cos}(ß-\alpha) = \frac{ax*ox+ay*oy}{|â| * |ô|}$$

Wobei der Winkel ß-α der von den Vektoren eingeschlosssene ist.

Für den Betrag eines Vektors (|â|) gilt nach dem Satz des Pythagoras:

$$|â| = \text{Sqr}\ (dax^2+day^2+daz^2)$$

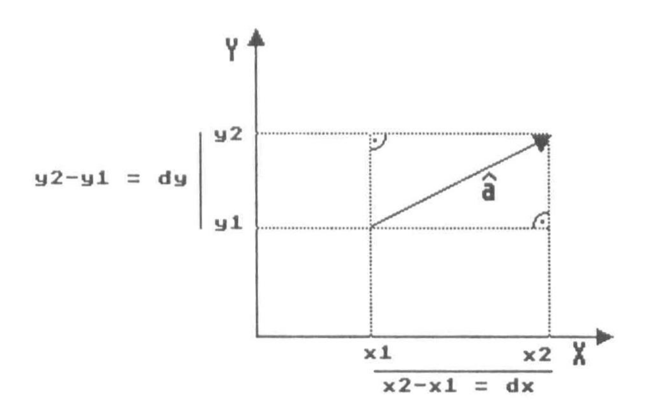


Abbildung 7.10

'dax' steht dabei für die Differenz aus X-Endkomponente minus X-Anfangskomponente eines Vektors. Ebenso natürlich 'day' und 'daz' für die Differenz der Y-, bzw Z-Komponenten.

Ersetzt man nun auch den Betrag der Vektoren in der Formel für den Cosinus und betrachtet zwei Vektoren im Raum (indem man einfach die Z-Koordinate mit einbezieht), erhält man für die Formel des Cosinus des eingeschlossenen Winkels zweier Vektoren:

$$\text{Cos}(ß-\alpha) = \frac{ax*ox+ay*oy+az*oz}{\text{Sqr}\ ((dax^2+day^2+daz^2)*(dox^2+doy^2+doz^2))}$$

7.4 Wenn Gerade und Ebene sich schneiden

Nun wollen wir uns den Schnittpunkten von Gerade und Ebene und von Gerade und Kugel widmen. Dies ist besonders beim Schattieren wichtig. Dort müssen wir den Punkt einer Fläche bestimmen, auf den ein Sehstrahl - repräsentiert durch eine Gerade - fällt.

Dabei war uns die 'Geradengleichung' des Sehstrahls, sowie die Körpergleichung des Körpers bekannt.

Für die Kugel brauchen wir nur für den Punkt auf der Oberfläche der Kugel die Geradengleichung einzusetzen und die einzelnen Komponenten zu betrachten, um den Schnittpunkt zu bestimmen:

$$\begin{matrix}(\hat{o}x & & rx & & Mx)^2 & & \\ (\hat{o}y & + & l*ry & + & My)^2 & = & r^2 \\ (\hat{o}z & & rz & & Mz)^2 & & \end{matrix}$$

Nehmen wir uns z.B. die X-Komponente vor, so gilt:

$$(l*rx + (\hat{o}x+Mx))^2 = r^2$$

Das 'l', für das die Gerade und die Kugel einen (Schnitt-) Punkt gemeinsam haben kann man durch Auflösen mit Hilfe der binomischen Formeln und eines Lösungsverfahrens für quadratische Gleichungen (z.B: PQ-Formel) bestimmen. Das 'l' ist der "Teleskop"-Faktor, mit dem der Richtungsvektor der Gerade multipliziert werden muß, um auf die Kugeloberfläche zu treffen. Ist jedoch die quadratische Gleichung unlösbar, so ist auch kein Schnittpunkt vorhanden.

Betrachtet man aber eine Gerade und eine Ebene, sieht die Sache mit dem Schnittpnukt schon etwas komplizierter aus. Hier muß man nämlich die Geraden- und die Ebenengleichung gleichsetzen:

$$\begin{matrix}\hat{o}x & & rx & & \hat{a}x & & r1x & & r2x \\ \hat{o}y & + & l*ry & = & \hat{a}y & + & a*r1y & + & b*r2y \\ \hat{o}z & & rz & & \hat{a}z & & r1z & & r2z\end{matrix}$$

Dann bildet man erst einmal die Gleichungen für jede Komponente und sorgt dafür, daß alle variablenbehafteten Elemente auf der linken Seite und die Absolutglieder auf der rechten Seite des Gleichheitszeichens stehen:

```
l*rx - a*r1x - b*r2x =  âx-ôx
l*ry - a*r1y - b*r2y =  ây-ôy
l*rz - a*r1z - b*r2z =  âz-ôz
```

Nun haben wir ein Gleichungssystem mit drei Variablen (l, a und b) und drei Gleichungen. Dieses 3x3 Gleichungssystem kann man nun mittels der sogenannten Determinanten lösen.

Vorher muß man jedoch eine Matrix aus den Gleichungen bilden. Dazu nimmt man alle Koeffizienten der Variablen und schreibt diese in eine Matrix hinein:

```
rx  -r1x  -r2x | âx-ôx
ry  -r1y  -r2y | ây-ôy
rz  -r1z  -r2z | âz-ôz
```

Dabei müssen die Koeffizienten vor gleichen Variablen untereinander stehen! Hat die Matrix also folgende Form

```
a11  a12 a13 | b1
a21  a22 a23 | b2
a31  a32 a33 | b3
```

(wobei a für die Koeffizienten und b für die Absolutglieder steht), können wir mit dem Bilden der Determinanten beginnen:

```
               | a11  a12 a13 |
D = det A =    | a21  a22 a23 |
               | a31  a32 a33 |

            =  a11 a22 a33 + a12 a23 a31 + a13 a21 a32
             - a13 a22 a31 - a11 a23 a32 - a33 a12 a21
```

Neben dieser 'Nennerdeterminate' müssen wir aber noch drei weitere Zählerdeterminanten bilden:

```
                 | b1  a12 a13 |
D1 = det A1 =    | b2  a22 a23 |
                 | b3  a32 a33 |

             =   b1 a22 a33 + a12 a23 b3 + a13 b2 a32
               - a13 a22 b3 - a23 a32 b1 - a33 a12 b2


                 | a11  b1 a13 |
D2 = det A2 =    | a21  b2 a23 |
                 | a31  b3 a33 |

             =   a11 b2 a33 + b1 a23 a31 + a13 a21 b3
               - a13 b2 a31 - a23 b3 a11 - a33 b1 a21

                 | a11  a12 b1 |
D3 = det A1 =    | a21  a22 b2 |
                 | a31  a32 b3 |

             =   a11 a22 b3 + a12 b2 a31 + b1 a21 a32
               - b1 a22 a31 - b2 a32 a11 - b3 a12 a21
```

Warum diese Determinanten 'Zähler-' und 'Nennerdeterminanten' heißen, erfahren Sie nun. Zur Lösung des Gleichungssystems müssen wir nämlich den Quotienten aus einer Zählerdeterminanten und der Nennerdeterminanten bilden.

Dabei hängt es von der 'Position' der Variablen im Gleichungssystem ab, welche Zählerdeterminante benutzt wird. In unserem Beispiel stand das 'l' an erster Stelle. Also wird der Qotient D1/D für diese Variable gebildet um das 'l' zu D bestimmen.

Die Lösungen für 'a' und 'b' erfolgen analog:

```
a=D2/D
b=D3/D
```

Das funktioniert natürlich nur dann, wenn die Nennerdeterminante ungleich 0 ist - denn die Division durch 0 ist nicht definiert. Bei D=0 hat das Gleichungssystem keine Lösung, oder anders gesagt: Ebene und Gerade haben keinen Punkt gemeinsam.

Wir haben nun einige Grundlagen der Mathematik angerissen und hoffen Ihnen Einiges verständlich dargelegt zu haben,für tiefergehende mathematische Betrachtungen wollen wir Sie allerdings auf die einschlägige Fachliteratur verweisen.

8. Tips und Tricks zur Bilderstellung

Nachdem Sie nun das Buch fast ganz durchgelesen haben, möchten wir Ihnen ein paar Tips geben, damit Sie leichter mit dem Programm zurechtkommen und überflüssige Fehler vermeiden.

Am Anfang ist der Entwurf. Eines der schwierigsten Probleme ist es, sich eine Szenerie auszudenken, die der Computer dann berechnen soll. Versuchen Sie es am Anfang mit wenigen Objekten. Zuviele Objekte machen das Bild unübersichtlich und verwirren den Betrachter nur. Ebenso sollten Sie nicht zu sehr ins Detail gehen, sondern sich erst einmal auf das Wesentliche beschränken. Einige Beispiele sind ja in diesem Buch abgebildet. Fällt Ihnen gar nichts ein, so können Sie ja auf die Objekte zugreifen, die Sie im Objekte-Ordner der Diskette finden.

Haben Sie eine halbwegs konkrete Vorstellung von dem, was Sie berechnen lassen wollen, so machen Sie sich am besten zuerst eine Skizze. Vorteilhaft ist es, wenn Sie diese auf Millimeterpapier zeichnen, da Sie dort die Maße der einzelnen Objekte besser ablesen können. Zeichnen Sie drei Ansichten Ihrer Szene, so wie diese auch im Editor angezeigt werden. Dabei sollten Sie sich auch schon die Materialkonstanten der einzelnen Objekte überlegen und diese in der Skizze festhalten, wobei Sie jedem Material eine Zahl (1..n) zuordnen.

Bei der Wahl der Farben sollten Sie beachten, daß das Bild weder zu grell noch zu farblos wird. Werfen Sie also nicht einfach mit allen möglichen Farben um sich, sondern wählen Sie einige Farben aus, die gut zusammen passen. Diese ordnen Sie dann den einzelnen Objekten zu, wobei Sie ja noch verschiedene Intensitäten verwenden können.

Die Schattenhelligkeit sollten Sie für alle Objekte auf etwa den selben Wert setzen (0 bis 0.25). Bei der Spiegelung kann weniger oft mehr sein. Verspiegeln Sie möglichst nur einzelne Objekte und nicht die ganze Szene, da man sonst hinterher vor lauter Spiegelungen nichts mehr erkennen kann. Setzen Sie nun noch-

mals in Gedanken die Szene zusammen und lassen Sie diese auf sich einwirken.

Fällt das Ergebnis zur Ihrer Zufriedenheit aus, so starten Sie den Editor und geben dort die einzelnen Objekte ein. Später, wenn Sie bereits mehr Übung im Ausdenken von Szenen haben, können Sie natürlich auf den Zwischenschritt mit der Papierskizze verzichten und die Daten direkt in den Editor eingeben.

Da Ihre Szene ja nicht allzu komplex seien sollte, setzen Sie im Editor für alle Objekte "sichtbar" auf "j", damit Sie auch genau sehen, was Sie eingegeben haben. Haben Sie alles eingegeben und für gut befunden, so speichern Sie dies ab und machen sich nun an die Materialkonstanten. Auch hier sollte die Eingabe keine Schwierigkeiten machen. Das Ganze muß nun wieder abgespeichert werden, und danach starten Sie den Tracer.

Als erstes stellt sich nun die Frage, in welcher Auflösung Sie das Bild berechnen lassen wollen. Wählen Sie den HAM-Modus, so haben Sie zweifelsohne die meisten Farben zur Verfügung. Allerdings kostet ein HAM-Bildschirm auch entsprechend Speicherplatz. Wenn Sie genug Speicher haben, so wählen Sie für den Anfang einen HAM-Bildschirm, da Sie hier die Farbpalette nicht so genau einzustellen brauchen.

Nachdem Sie nun Ihre Objekte und Materialien geladen haben, müssen Sie sich nun überlegen, aus welchem Blickwinkel Sie die Szene betrachten wollen. Legen Sie der Einfachheit halber den Hauptpunkt mitten in Ihre Szene hinein. Nun betrachten Sie nochmals Ihre Skizze und überlegen sich, von wo aus Sie die Szene betrachten wollen. Ermitteln Sie die Koordinaten Ihres Standpunktes. Beachten Sie dabei, daß sich Ihr Standpunkt nicht zu nah an den Objekten befinden sollte, da die perspektivischen Verzerrungen sonst sehr groß werden.

Geben Sie nun Ihren Standpunkt als Projektionspunkt ein. Ist Ihnen kein vernünftiger Standpunkt eingefallen, so wählen Sie versuchsweise den Punkt (-1000,-1000,1000). Drücken Sie nun die Space-Taste, um sich das Drahtmodell anzeigen zu lassen. Wahrscheinlich ist dies nur relativ klein auf dem Bildschirm zu

sehen. Drücken Sie also "V", um das Bild vergrößern zu lassen. Die Objekte sollten den Bildschirm ausfüllen, nicht aber direkt bis zum Rand reichen. Haben Sie spiegelnde Flächen definiert, so wählen Sie den Ausschnitt etwas größer, damit die Spiegelungen auch noch mit aufs Bild kommen.

Probieren Sie nun solange herum, bis Ihnen die Darstellung gefällt. Dann schreiben Sie sich als allererstes die Werte auf, die Sie eingegeben haben, also Hauptpunkt, Vergrößerungsfaktor und entweder Projektionspunkt oder Winkel und Abstand, damit Ihr Bild später mal reproduzierbar ist.

Nun gilt es, die Position der Lichtquelle festzulegen. Empfehlenswert ist hier eine Position "seitlich" vom Betrachter, so daß man die Schatten gut sehen kann. Liegt der Projektionspunkt beispielsweise bei (-1000,-1000,1000), so könnte die Lichtquelle bei (-1000,1000,1000) liegen, wenn die Schatten nach rechts fallen sollen. Wollen Sie kürzere Schatten, so vergrößern Sie einfach die Z-Koordinate der Lichtquelle. Für längere Schatten verkleinern Sie diese entsprechend.

Auch die Position der Lichtquelle schreiben Sie sich auf. Nun drücken Sie F9, um die Parameter für das Schattieren zu initialisieren. Wählen Sie den ganzen Bildschirm als Ausschnitt, und geben Sie daran anschließend die Position der Lichtquelle ein. Als Farbe der Lichtquelle wählen Sie (1,1,1), was einer weißen Lichtquelle entspricht.

Für Punktbreite und -höhe geben Sie jeweils ein Zwanzigstel der Bildschirmbreite und -höhe an, also 16 und 10 für einen 320x200-Bildschirm. Da Sie erst einmal nur einen groben Überblick über Ihre Szene gewinnen wollen, reichen diese dicken Pixel aus, vor allem, da die Berechnung auch nur ein vierhundertstel der Zeit kostet.

Lassen Sie dann das Bild berechnen (F10). Stoppen Sie dabei die Zeit, die zwischen dem Setzen des ersten und des letzten Punktes liegt. Nachdem das Bild fertig ist, können Sie dann ausrechnen, wie lange der Computer für das Bild in voller Auflösung be-

nötigen würde, indem Sie die Zeit mit Punktbreite mal Punkthöhe multiplizieren.

Sind Sie kurzsichtig, so sind Sie nun im Vorteil. Setzen Sie Ihre Brille ab, und treten Sie soweit vom Bildschirm zurück, daß die Rechtecke der einzelnen Punkte so gerade eben verschwinden. Nun können Sie bereits einen (groben) Eindruck vom späteren Bild ergattern. Nicht Kurzsichtige müssen dementsprechend weiter zurücktreten oder sich eine Brille für Kurzsichtige aufsetzen.

Nun sehen Sie auch, ob die Farben so herauskommen, wie Sie sich das gedacht haben. Es ist sehr wichtig, daß Sie die Farbpalette optimal einstellen, auch bei HAM-Bildern. Betrachten Sie dazu das Bild, das Sie bereits in grober Auflösung haben ausrechnen lassen, und merken Sie sich die Farben, die darin vorkommen. Stellen Sie nun die Farbpalette so ein, daß für jede vorkommende Farbe möglichst viele verschiedene Helligkeitsstufen vorhanden sind. Je weniger Farben Sie verwendet haben, um so mehr Abstufungen können Sie nun einstellen.

Beim HAM-Modus sollten Sie die Farbpalette so einstellen, daß bei jedem plötzlichen Übergang von einer Farbe zu einer anderen, möglichst beide Farben direkt in der Palette enthalten sind. Beachten Sie dabei aber, daß wegen der Besonderheiten des HAM-Modus ein Übergang von Schwarz nach rot, grün oder blau überhaupt keine Probleme macht, während ein Übergang von schwarz nach grau nicht direkt vollzogen werden kann, wenn dieser Grauton nicht in der Palette vorkommt.

Haben Sie alle Farben optimal eingestellt, so lassen Sie das Bild nochmals berechnen. Wählen Sie die Punktgröße jetzt ruhig etwas kleiner (halb so groß wie eben), damit Sie mehr Einzelheiten erkennen können. Sind Sie mit dem Ergebnis zufrieden, so können Sie sich jetzt nochmal überlegen, ob Sie nicht lieber einen anderen Darstellungsmodus wählen wollen. Wenn Sie dies wollen, so sollten Sie sich ebenfalls die Belegung der Farbpalette aufschreiben, damit Sie gleich nicht schon wieder herumexperimentieren müssen.

Bevor Sie dann (endlich) das Bild in voller Auflösung ausrechnen lassen, sollten Sie sich überlegen, wann Ihr Computer damit fertig ist. Nicht die schlechteste Idee ist es, das Bild über Nacht ausrechnen zu lassen. Dies hängt allerdings von der Lautstärke Ihres Lüfters ab und von der Tiefe Ihres Schlafes:

```
überNacht = Tiefe(Schlaf) / Lautstärke(Lüfter).
```

Wenn der Computer mehrere Tage zum Rechnen braucht, wie es bei (sehr) komplexen Bildern schon mal vorkommen kann, so können Sie das Bild auch teilweise ausrechnen lassen, abspeichern und später weiter ausrechnen lassen, indem Sie das Bild laden (wodurch auch die Farbpalette richtig eingestellt wird), die notierten Parameter einstellen, bei Auschnitt aber "Ya-Ye" wählen und den noch nicht schattierten Bereich des Bildes angeben.

Haben Sie genug Erfahrungen gewonnen, so können Sie das Programm auch erweitern, wie es bereits in den Ausblicken beschrieben wurde. Sie können dazu am Anfang das Programm aber auch ganz speziell für ein bestimmtes Bild ändern, wie wir es auch bei den Bildern mit den Relief-Böden gemacht haben. Zwar mußten wir dafür eine spezielle Programmversion erstellen, der programmtechnische Aufwand betrug aber nur etwa eine Bildschirmseite. Für den Reliefboden haben wir in BestimmeHell eine kleine Abfrage eingebaut, ob der Körper die Nummer Eins trägt (welche die Nummer des Bodens war) und dann Nx und Ny in Abhängigkeit von Sx und Sy verändert (SIN bzw Fallunterscheidung). Danach mußte der Normalenvektor freilich wieder normiert werden, aber das entsprechende Programmstück brauchten wir ja bloß zu kopieren.

Ebenso einfach wäre es gewesen, ein Muster auf die Fläche zu werfen, indem der Wert von K(n,0,2) in Abhängigkeit von Sx und Sy verändert wird. Natürlich können Sie das Programm auch so erweitern, daß Sie eine allgemeingültige Lösung dafür schaffen, so daß Sie auch auf eine Kugel ein Muster (oder ein Relief) werfen können. Ihrer Phantasie sind dabei, wie bereits in den Ausblicken erwähnt, keine Grenzen gesetzt, außer der Rechenzeit und der Häufigkeit eines Stromausfalles in Ihrer Gegend.

Anhang A - Der Modularaufbau des Tracers

1. Initial-Routinen

DECLARE FUNCTIONs
InitMat
InitMusrterS
InitMusterX
RasterInit
OpenGfx
InitSetPoint
CloseSetPoint

2. Trace-System-Routinen

Ausgabe
EmptyBuffers
Warte
Box
DialogBox
DoDialog
Rubberbox
Noop
MakeMenu
DeleteMenu
MessageEvent
KeyEvent
FehlerBehandlung
EditorAufrufen
CloseIt
Quit
BildLoeschen
Scron
Scroff
FarbPalette

GetString
GetReal
GetInt
GetKey
SetCursor s. Editor
PutString
PutReal
conrealstr
constrreal

FormInputString
FormInputInt
FormInput
Directory

3. Drahtmodell-Routinen

Projektion

ZeichneEbene
ZeichneDreieck
ZeichneViereck
ZeichneKreis
ZeichneKreisSegm
ZeichneKreisTeil
ZeichneKugel
ZeichneZylinder
ZeichneZylinderSegm
ZeichneKegel
ZeichneEllipsoid

ZeichneAlle
ZeichneNeu

WievielEcken
Vergrößern

4. Eingabe-Routinen

Initial
InitialP

InputP
InputH
InputDph
InputWinkel

5. Schattieren-Initialisieren

InitParameters

Transform
InitSchattiere

TransformMM
MinMaxTest
InitMinMax

NormWinkel
DeltaSum3
DeltaSum4
Getxyzlq
Getxyzlqk
DistTest
ArcTan
CalcABlq

MMlq3Test
MMlq4Test
MMlqkTest
MMlqKugel
MMlqZylinder
MMlqKegel
MMlqEllipsoid

InitMinMaxlq

6. Schattieren

SchnittpunktEbene
SchnittpunktDreieck
SchnittpunktViereck
SchnittpunktKreis
WinkelInterwall

SchnittpunktKreisSegm
SchnittpunktKreisTeil
SchnittpunktKugel
BasisTrans

SchnittpunktZylinder
SchnittpunktZylinderSegm
SchnittpunktKegel
SchnittpunktEllispsoid

WelcherKoerper
BestimmeHell

StandardFill
ExtendedFill
SetPoint

Reprojektion
Berechnepunkt

Schattiere

7. Service-Modul

Speichern
Laden
Mergen
FeldInit

BildSpeichern
BildLaden

Hardcopy

Himmel
Fhg
Hintergrund

Info
Help

Anhang B - Fehlermeldungen des Tracers

Während des Programmablaufs können verschiedene Fehler auftreten. Die häufigsten Fehler sind wohl die Ein-/Ausgabefehler, die dann auftauchen, wenn z.B. versucht wurde, auf ein nicht vorhandenes File zuzugreifen, oder wenn die Diskette einen Schreib-/Lesefehler hat.

Solche Fehler werden in der Tracer-Routine 'Fehlerbehandlung' bearbeitet. Auch durch falsches Zusammensetzen der Module entstandene Fehler werden hier abgefangen.

Fehler bei der Übertragung der IFF-Bilder von und auf Diskette werden aber nicht über diese Routine bearbeitet. 'Fehlerbehandlung' wird durch 'ON ERROR' aktiviert. 'BildLaden' und BildSpeichern' benutzen für die Diskettenaktivitäten aber nicht die BASIC Befehle, sondern die sogenannten Library-Befehle. Hier auftretende Fehler müssen also getrennt abgefangen werden.

Diese Fehler (z.B. '...-Chunk konnte nicht abgespeichert werden!!!') lassen sich allerdings meistens auf volle Disketten zurückführen.

Fehlermeldungen beim Initialisieren zu Beginn des Programms ('Kein freier Chip-Mem mehr!!!', o.ä.) lassen auf nicht ausreichenden Speicherplatz schließen. Hier hilft meist nur ein 'abspecken' der Programme, wenn schon Tricks wie das möglichst kleine 'AmigaDOS'-Window ohne Wirkung blieben. Sie müssen also weniger wichtige Routinen (z.B. 'Hintergrund' und Co) weglassen bzw. löschen. Leider kann es auch passieren, daß sich der Rechner oder vielmehr das AmigaBASIC aufgrund Speichermangels 'verabschiedet'. Besonders bei einem großen Array-K (ca. 200 und mehr Elemente) kann dies bei Maschinen mit nur 500 KB auftreten.

Aber auch während der Initialisierung beim Schattieren können Fehler auftreten: Wenn nämlich die Basis eines Körpers gar keine Basis ist, wird Ihnen dies mitgeteilt. (Eine Basis besteht aus drei linear unabhängigen Vektoren. Ein Vektor läßt sich also

nicht als Vielfaches der anderen beiden Vektoren darstellen). Dieser Fehler tritt dann auf, wenn die Vektoren a,b und c eines Ellipsoids, Kegels oder Zylinders keinen Raum aufspannen, also komplanar (Ebene) oder kollinear (Gerade) sind.

Wenn sich also einer der drei Vektoren a,b oder c mittels der zwei verbleibenden Vektoren (z.B. a = x*b + y*c) darstellen läßt, wird nur eine Ebene oder Gerade und kein Raum aufgespannt!

Stichwortverzeichnis

2x2-Unterdeterminanten 42
3-D nach 2-D 83
3-D/2-D-Transformation 92
4x4-Determinanten 47

Ableitung 57
Abspeichern 178, 230
Abstand 59, 232
Addieren 248
Amiga-Menü 225
Anklicken 121
Anzahl Ecken 232
AnzahlMat 34
ASCII-Files 209
Assemblerroutine 72
Auflösung 224, 262
Ausblicke 239
Ausdrucken 230
Ausgabeprozedur 138
Ausgaberoutinen 135
Ausschnitt 118, 152
AUSSCHNITT 220
Ausschnittfensters 93
Ausschnittwahl 233

B-Splines 240
BasisTrans 49
Basistransformation 47, 50
Benutzer-Screen 160
BerechnePunkt 40, 67
Berechnungszeit 81
BestimmeHell 60
Betrag 255
Betriebssystemprozeduren 145
Bezierfunktionen 240
Bezugspunkt 147
Bilderstellung 261

Bildschirmausschnittes 198
Bildschirmkoordinaten 92, 152
Blau-Anteil 34

Darstellungsmodus 225
Datei-Menü 225
Datenaustausch 169
Datenein- und -ausgabe 120
Datenfile 150
Datenmanipulation 147
Datenoperationen 143
Datensatz 213
Datenstruktur 32
Default-Farbwerte 167
Defaultwerte 236
Definitionsbereich 24
Determinanten 42, 257
Differenzvektoren. 248
Direktmodus 209
DISK 223
Diskettenoperationen 150, 223
Diskettenverzeichnis 204
Diskettenwechsel 204
Display-Screen 160
Dph 36
Drahtmodell 83, 117, 236, 237
Draufsicht 117
Drehung 90
Dreidimensionale Geometrie 23
Dreieck 25, 31, 43
Dreiecksfläche 26

Ebene 23, 41, 256
Ebenengleichung 23, 26
Editieren 224
Editor 115, 121, 150, 159
Editor-Menü 232
Einfallswinkel 56
Eingabe 123, 211
Eingabeprozedur 129, 132

Einheitskreis 27
Einheitsvektor 25
Einlese-Operationen 127
Einschränkung 24, 26, 29, 34
Ellipse 28
Ellipsengleichung 57
Ellipsoid 29, 31, 52, 58
Ellipsoidausschnitt 32
Entwurf 261
Erweiterungen 239
Extra Halfbrite 225

Farbe der Lichtquelle 40
Farben 69, 261
Farbpalette 73, 231, 264
Fehlerbehandlung 207
Fehlermeldung 151
Flächen 99
Fließender Hintergrund 229

Gadgets 121, 152, 153
Gerade 23, 40, 256
Geradengleichung 41, 47, 250, 256
Geradenparameter 41
Gleichungssystem 257
Grad 95
Grafik 138
Grafikobjekte 139
GRAPHICS-LIBRARY 124
Grün-Anteil 34
Grundobjekte 26, 29, 32, 239

HAM-Modus 69
Hardcopy 230
Hardcopy-Routine 178
Hauptprogramm 209
Hauptpunkt 36, 84, 232
Hausaufgabe 32
Helligkeit 56, 60, 157

Help 237
Hidden-Line 17, 117
Hidden-Surface 17
Hilf() 43
Himmel 229
Hintergrund 40, 114, 228
Hold and Modify 225

IFF-Format 174
Indirekte Beleuchtung 23
InitMinMax 76
InitMinMaxLq 78
InitSchattiere 74
InitSetPoint 72
Intervallgrenzen 28

K() 32
Kegel 29, 31, 52, 58
Kegelausschnitt 31
Kegelspitze 58
Kegelstumpf 31
Komplanar 29
Konvertierungsprozeduren 132
Konvertierungsroutinen 128, 129
Koordinatengleichungen 47, 50
Kopieren 120, 222
Körper 99
Körperdaten 213
Körpergleichung 50
Korrekturen 120
Korrekturfaktors 138
Kreisausschnitt 28, 44
Kreisbogen 28, 45
Kreisdarstellung 112
Kreisfäche 44
Kreisgleichung 27
Kreisringe 28
Kugel 23, 29, 33, 46
Kugelgleichung 29
Künstliche Wirklichkeit 36

Laden 226
Längsschnitt 31
Laufzeit 73
Licht 117
Lichtbrechungen 23
Lichtquelle 18, 20, 63, 77, 235, 263
Lichtreflex 60
Lichtstrahlen 18, 22, 55, 59
Linear 23
Load-Funktion 121
Loadlist 223
Loadmat 223
Löschen 222

Malprogramm 174
Manipulationen 120
Maschinenroutine 210
Mat 34
MAT.EDITOR 222
MAT.EDITOR-Funktion 220
Material 222
Materialdaten 158
Materialeditor 157, 159, 213, 221
Materialelement 157
Materialien 213, 228
Materialkonstanten 33, 34, 173
Materialnummer 221
Mathematik 247
Maus 152, 212
Mausabfrage 152
Mauseingabe 131
Mauskoordinaten 152
Mausposition 212
Maustaste 152
Mausvektor 131
Mauszeiger 153
MaxAnzahl 32
Melonenscheibe 32
Menüauswahlen 195

Menüs ... 134
Menüsteuerung ... 192, 238
Menüzeilen ... 189
Mergen ... 172, 209, 227
Multitaskingbetrieb ... 236
Muster ... 70, 229, 242

Nennerdeterminante ... 42, 50, 74, 258
Neue Farben ... 231
Neustart ... 206, 231
New ... 227
NOP ... 195
Normal ... 225
Normalenvektor ... 36, 55, 74, 240
Normiert ... 63

Oberfläche ... 55
Objektdefinitionen ... 169
Objekteditor ... 160, 207
OK-Symbols ... 158
Ortsvektor ... 248, 250
OSSetPoint& ... 40

Parallelogramm ... 24, 44
Parameter'-Menü ... 232
Perspektive ... 90
Perspektivtransformation ... 92
Plastizität ... 63
Polarkoordinaten ... 28, 72, 77
PROFIMAT ... 72
Programmende ... 231
Programmmodule ... 209
Projektion ... 39
Projektionsebene ... 77, 84
Projektionspunkt ... 36, 41, 83, 232
Proportionalvergrößerung ... 233
Punkt-Richtungs-Form ... 41
Punktbreite ... 235
Punkthöhe ... 235

Quadrat 25
Quadratisches Gleichungssystem 46
QUIT 220

Radiant 95
Radius 33
RasterInit 73
RASTPORT-Datenstruktur 123
Raum 29
Raumkoordinaten 152
Ray-Tracing 17
Rechenzeit 112, 117
Rechteck 25
Reflektion 21
REFRESH 220
Rekursiv 22
Reprojektion 39
Richtungs 248
Richtungsvektor 41, 54, 250
Rot-Anteil 34
Rotation 120, 147, 223
Rotationskörper 241

Save 121
Savelist 223
Savemat 223
Schatten 20, 46, 63, 77, 117, 157
Schattenhelligkeit 34, 64, 213, 261
Schattieren 224
Schattierung 20
Schnittpunkt 40, 55, 63, 91, 256
Schnittpunktsberechnung 74
Schwelle 46, 64
Sehstrahl 19, 20, 22, 36, 54, 59, 66, 75
Seitenansicht 117
Senkrecht 27, 29, 57
SetPoint.ASM 72
SetPoint.B 73
Shortcuts 195, 237
SHOW 220

Skalare 23
Skalarprodukt 252
Smooth-Shading 240
Speichern 226
Speicherverwaltungsroutinen 145
Spektrum 23
Spiegelung 21, 40, 46, 66, 117, 261
Spiegelungsfaktor 35, 157, 213, 221
Standpunkt 262
Steigung 57
Streulicht 23, 64
Subtrahieren 248
Superrechner 81

Tastatur 237, 238
Tastendrücke 195
Tracer 83
Transformationen 222
Translation 223
Transparenz 21, 243
Transparenzfaktor 35

Ungenauigkeit 46, 64

Vektor 23, 33
Vektorprodukt 56
Vektorrechnung 247
Verbesserungen 239
Vergößerung 120, 147, 149, 223
Vergößerungsfaktor 152
Vergrößern 191
Vergrößerungsart 192
Vergrößerungsfaktor 93, 220, 233
Verschieben 120
Verschiebung 147, 149
Verspiegelt 21
Vier Schnittpunkte 239
Vorderansicht 117

Wahrnehmung 36
WelcherKoerper 52
Winkel 55, 253
WinkelIntervall 45
Würfel 25

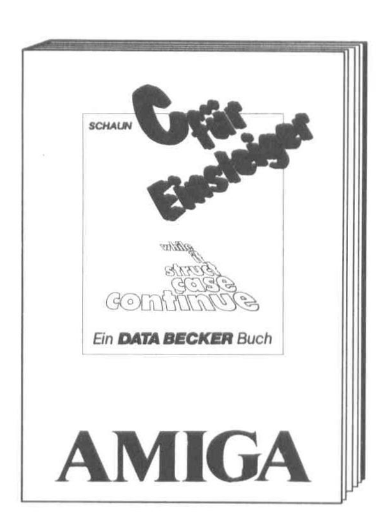
X-Komponente 33
X-Korrekturfaktor 152

Y-Komponente 33

Z-Komponente 33
Zähler 258
Zeichenfarbe 139
Zeichengeschwindigkeit 140
Zeichenkette 123
Zeichenprogramm 181
Zeichenprozedur 142
Zeichenroutine 140, 143
Zeichne'-Menü 234
Zeit 264
Zwei Schnittpunkte 46
Zylinder 29, 47, 56
Zylinderausschnitt 31, 51
Zylindergleichung 47

Bücher zum Amiga

Wer auf dem Superrechner AMIGA schnell in die Supersprache C einsteigen will, der findet in diesem Buch den schnellen Einstieg: Mit „C an einem Wochenende“. Dieser Einführungskurs löst die Anfangsschwierigkeiten und führt gleichzeitig in die Bedienung der beiden wichtigsten vorhandenen Compiler ein. Anschließend wird der gesamte Sprachumfang mit vielen Beispielen erläutert und der Umgang mit den Routinen der C-Bibliotheken erklärt.

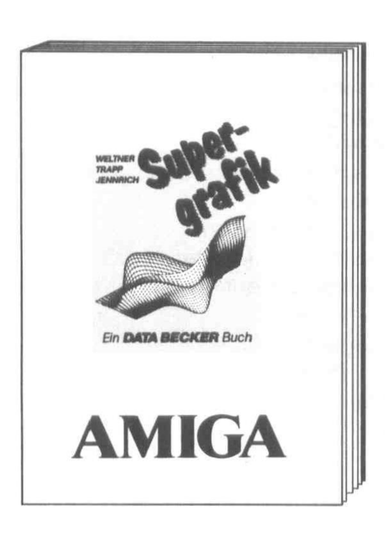


Aus dem Inhalt:

- Das Besondere an C
- Bedienung eines C-Compilers
- Das erste Programm
- Der Sprachumfang von C (Schleifen, Bedingungen, Funktionen, Strukturen)
- Die speziellen Fähigkeiten von C
- Die wichtigsten Routinen der C-Bibliotheken
- Ein-/Ausgabe
- Tips und Tricks zur Fehlersuche
- Arbeit mit Lattice C und dem Aztek Compiler

Schaun
C für Einsteiger
254 Seiten, DM 39,–
ISBN 3-89011-107-6

Der Amiga ist eine tolle Grafik-Maschine. 4096 Farben gleichzeitig, 640 × 400 Punkte Auflösung, Sprites, Bobs und die Geschwindigkeit des Blitters begeistern einfach. Das AMIGA SUPERGRAFIK-Buch zeigt Ihnen, wie Sie diese tollen Features in den Griff bekommen. Dabei ist es gleichgültig, ob Sie mit BASIC einsteigen, über die Amiga-Libraries die schnellen Routinen von BASIC aus nutzen oder komplette Programme in C schreiben wollen: Dieses Buch zeigt mit vielen Beispielprogrammen, wie man die Grafik des Amiga programmiert.

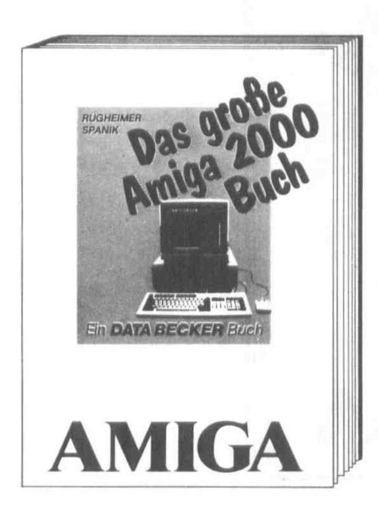


Aus dem Inhalt:
- Die Grafik-Befehle des AmigaBASIC (Punkt, Linie, Kreis, Rechteck, Muster, Flächen füllen)
- Laden und Speichern von Grafiken (IFF-Format)
- Sprites, Bobs, Animation
- CAD durch 1024 × 1024-Superbitmap
- Verschiedene Zeichensätze und Schriftarten in BASIC
- Neue Möglichkeiten von BASIC aus durch Zugriff auf die Libraries und Spezial-Chips (4096 Farben gleichzeitig, farbige Muster, Hardcopy von Screens und Windows)
- Grafikprogrammierung von C aus (Punkt, Linie, Rechtecke, Polygone, Farben)
- Die Routinen der Grafik-Bibliothek nutzen
- Das Animationssystem des Amiga (Sprites, Bobs, AnimObs)
- Programmierung von Copper und Blitter (Rasterzeilen-Interrupt, blitzschnelles Kopieren)
- Komplette Beschreibung des Amiga-Grafiksystems (View, Viewport, RastPort, Aufbau der Bitmaps, Screens, Windows)

Weltner/Trapp/Jenrich
Supergrafik
686 Seiten, DM 59,–
ISBN 3-89011-254-4

Bücher zum Amiga 2000

Der Amiga 2000 wird immer beliebter. Kunststücks hat er doch wirklich das Zeug zum Traumcomputer. Besonders für diejenigen, die einen Amiga wollen, aber auch einen PC brauchen. Das einzige, was den Amiga-2000-Besitzern bisher fehlte, war die passende Literatur. Das hat sich mit diesem Buch geändert.



Aus dem Inhalt:

- Amiga-Grundlagen leichtgemacht
- So stellt man die Akku-Uhr
- Vom richtigen Umgang mit AmigaDOS
- Software und was man damit machen kann
- Software-Installationstips für die Harddisk
- Einbau der PC-Karte
- Speicher erweitern – aber richtig
- Einbau und Einrichten von PC- und Amiga-Harddisk
- Wie man Kickstart wieder ins RAM bringt
- Ton auch für den PC

Rügheimer/Spanik
Das große Buch zum Amiga 2000
Hardcover, 684 Seiten, DM 59,–
ISBN 3-89011-199-8

DAS STEHT DRIN:

Der Amiga hat ausgezeichnete Grafikfähigkeiten, aber vielen Amiga-Besitzern fällt es schwer, diese Fähigkeiten auch in eigenen Programmen auszuschöpfen – besonders, wenn es um dreidimensionale Darstellungen geht. Mit diesem Buch wird das einfach, weil nicht nur die entsprechenden Algorithmen ausführlich erklärt, sondern auch gleich die fertigen Lösungen angeboten werden.

Aus dem Inhalt:

- Grundlagen des Ray-Tracings
- Eingabe der dreidimensionalen Objekte über komfortablen Objekteditor
- Materialbestimmung über Materialeditor
- Automatische Berechnung in verschiedenen Auflösungen
- Alle Amiga-Auflösungen können genutzt werden (Lowres, Hires, Interlace, Hold and Modify)
- Verschiedene Lichtquellen, beliebiger Blickpunkt
- Grafiken können anschließend im IFF-Format gespeichert und dadurch in die gängigen Malprogramme eingelesen werden
- Mathematische Grundlagen auch für Nichtmathematiker
- Die Diskette im Buch enthält nicht nur alle Routinen als editierbare BASIC-Versionen, sondern auch die compilierten Fassungen, die deutlich schneller sind

UND GESCHRIEBEN HABEN DIESES BUCH:

Bruno Jennrich ist begeisterter Grafik/3-D-Programmierer. Mit dem Informationspaket „Supergrafik" zum Amiga hat er sein Know-how unter Beweis gestellt. Peter Schulz ist Autor von „Profimat Amiga". Wenn sein Amiga einmal nicht mit dem Assemblieren von Maschinensprache-Programmen beschäftigt ist, läßt Peter Schulz ihn vorwiegend dreidimensionale Bilder berechnen. Andreas Massmann stieg gleich mit einem Amiga 2000 in die Amiga-Welt der Grafik ein. Dadurch hat er auch bei den höchsten Auflösungen der Bilder nie mit Speicherproblemen zu kämpfen.

